DL LAB 5: CNN with Regularization Techniques

Name : Mahesh Jagtap Reg No 24MCS1017

Choose a suitable dataset. (Any one among, plant, medical, dental, marine, manufacturing, domain data)

1. Try to implement CNN and study the problem of overfitting
2. Apply L1 and L2, and write the inference
3. Apply Drop out and write the inference
4. Apply Data augmentation and write the inference
5. Apply Early Stopping and write your inference

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```


# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize the data to the range [0, 1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)

x_train = x_train[:3000]
y_train = y_train[:3000]

x_test = x_test[:500]
y_test = y_test[:500]

# Print the shape to verify
print(x_train.shape)  # (3000, 32, 32, 3)
print(y_train.shape)  # (3000, 10)
print(x_test.shape)   # (500, 32, 32, 3)
print(y_test.shape)   # (500, 10)
```

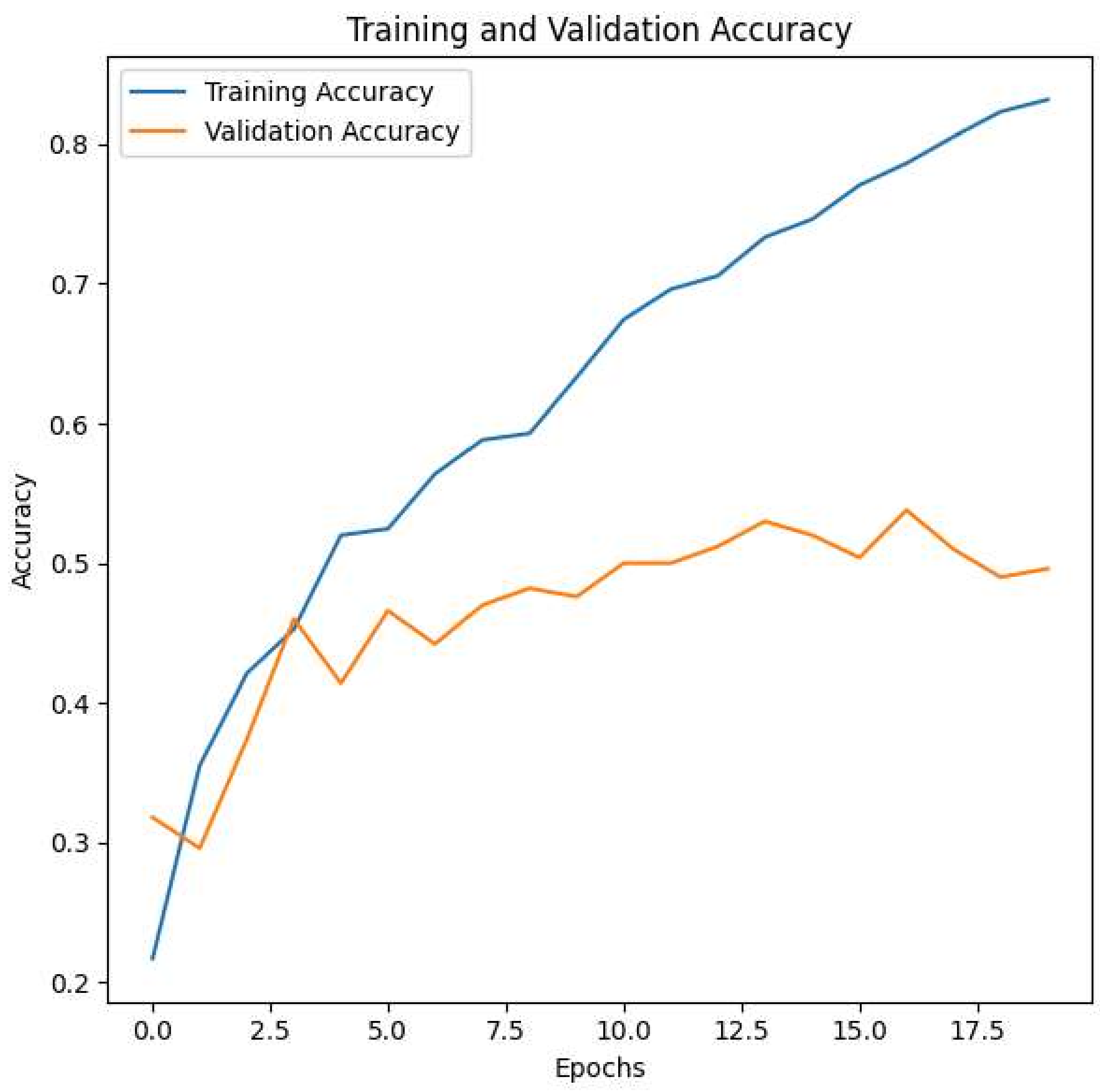
```
(3000, 32, 32, 3)
(3000, 10)
(500, 32, 32, 3)
(500, 10)
```

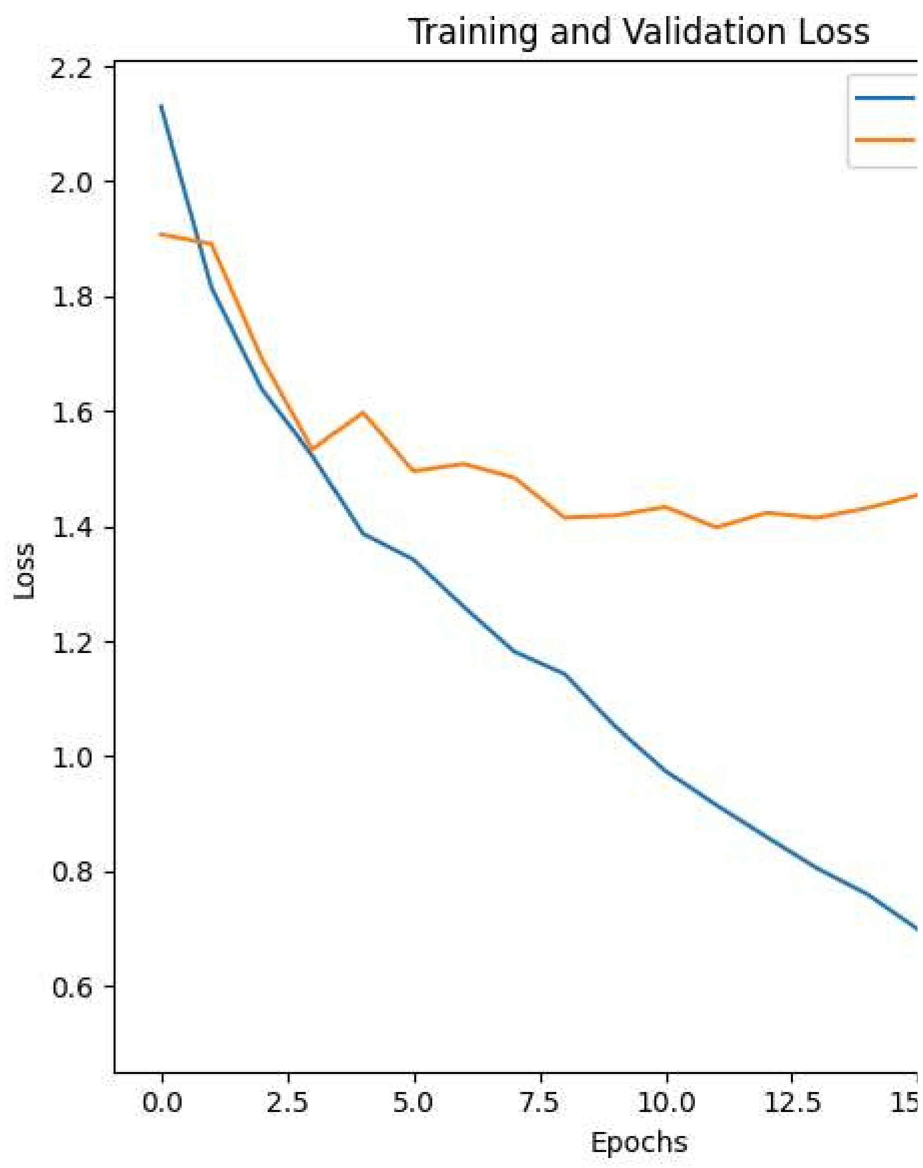
```
def create_model():
    model = keras.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(10, activation='softmax')
    ])
    return model

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

# Change epochs to 10
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))
```

```
Epoch 1/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 40ms/step - accuracy: 0.1609 - loss: 2.2353 - val_accuracy: 0.3180 - val_loss: 1.9076
Epoch 2/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.3269 - loss: 1.8474 - val_accuracy: 0.2960 - val_loss: 1.8909
Epoch 3/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.3940 - loss: 1.6935 - val_accuracy: 0.3740 - val_loss: 1.6913
Epoch 4/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.4520 - loss: 1.5218 - val_accuracy: 0.4600 - val_loss: 1.5339
Epoch 5/20
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.5256 - loss: 1.3911 - val_accuracy: 0.4140 - val_loss: 1.5971
Epoch 6/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.5135 - loss: 1.3584 - val_accuracy: 0.4660 - val_loss: 1.4962
Epoch 7/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.5814 - loss: 1.2409 - val_accuracy: 0.4420 - val_loss: 1.5085
Epoch 8/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.5932 - loss: 1.1830 - val_accuracy: 0.4700 - val_loss: 1.4847
Epoch 9/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.5996 - loss: 1.1396 - val_accuracy: 0.4820 - val_loss: 1.4151
Epoch 10/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.6309 - loss: 1.0497 - val_accuracy: 0.4760 - val_loss: 1.4187
Epoch 11/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 39ms/step - accuracy: 0.6843 - loss: 0.9638 - val_accuracy: 0.5000 - val_loss: 1.4336
Epoch 12/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.7086 - loss: 0.8829 - val_accuracy: 0.5000 - val_loss: 1.3981
Epoch 13/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.7221 - loss: 0.8395 - val_accuracy: 0.5120 - val_loss: 1.4233
Epoch 14/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.7439 - loss: 0.7898 - val_accuracy: 0.5300 - val_loss: 1.4148
Epoch 15/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 52ms/step - accuracy: 0.7659 - loss: 0.7284 - val_accuracy: 0.5200 - val_loss: 1.4317
Epoch 16/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.7890 - loss: 0.6726 - val_accuracy: 0.5040 - val_loss: 1.4541
Epoch 17/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.7922 - loss: 0.6402 - val_accuracy: 0.5380 - val_loss: 1.4977
Epoch 18/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 46ms/step - accuracy: 0.8237 - loss: 0.5784 - val_accuracy: 0.5100 - val_loss: 1.4732
Epoch 19/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 40ms/step - accuracy: 0.8277 - loss: 0.5503 - val_accuracy: 0.4900 - val_loss: 1.5923
Epoch 20/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 47ms/step - accuracy: 0.8452 - loss: 0.5122 - val_accuracy: 0.4960 - val_loss: 1.6285
```

```python
import matplotlib.pyplot as plt

# Plot accuracy
plt.figure(figsize=(12, 6))

# Plot training accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```





Training and Validation Accuracy:

The training accuracy increases steadily from around 0.2 to above 0.8 over the epochs, showing that the model is learning the training data well.

However, the validation accuracy (orange line) starts around 0.3, rises to about 0.5, and then fluctuates around this value without significant improvement. This lack of improvement suggests that the model isn't performing well on unseen data.

Training and Validation Loss:

The training loss decreases steadily from above 2.0 to below 0.6, further indicating that the model is fitting the training data well.

In contrast, the validation loss (orange line) starts around 2.0, drops to about 1.4, and then fluctuates without a clear downward trend. This discrepancy suggests that the model isn't generalizing well to new data.

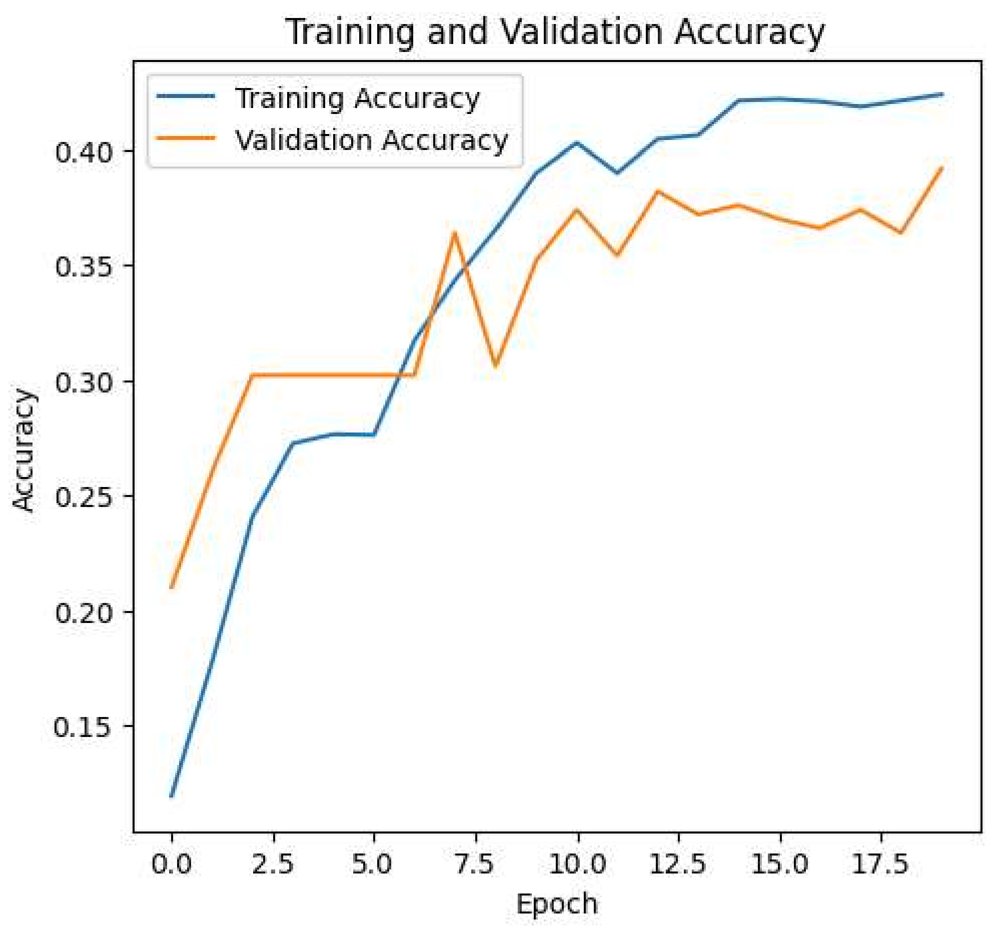
Inference: The model is overfitting. While it performs well on the training data, it struggles to maintain high accuracy and low loss on the validation data, indicating poor generalization. This overfitting could be addressed by techniques such as using more data, applying regularization methods, or early stopping.

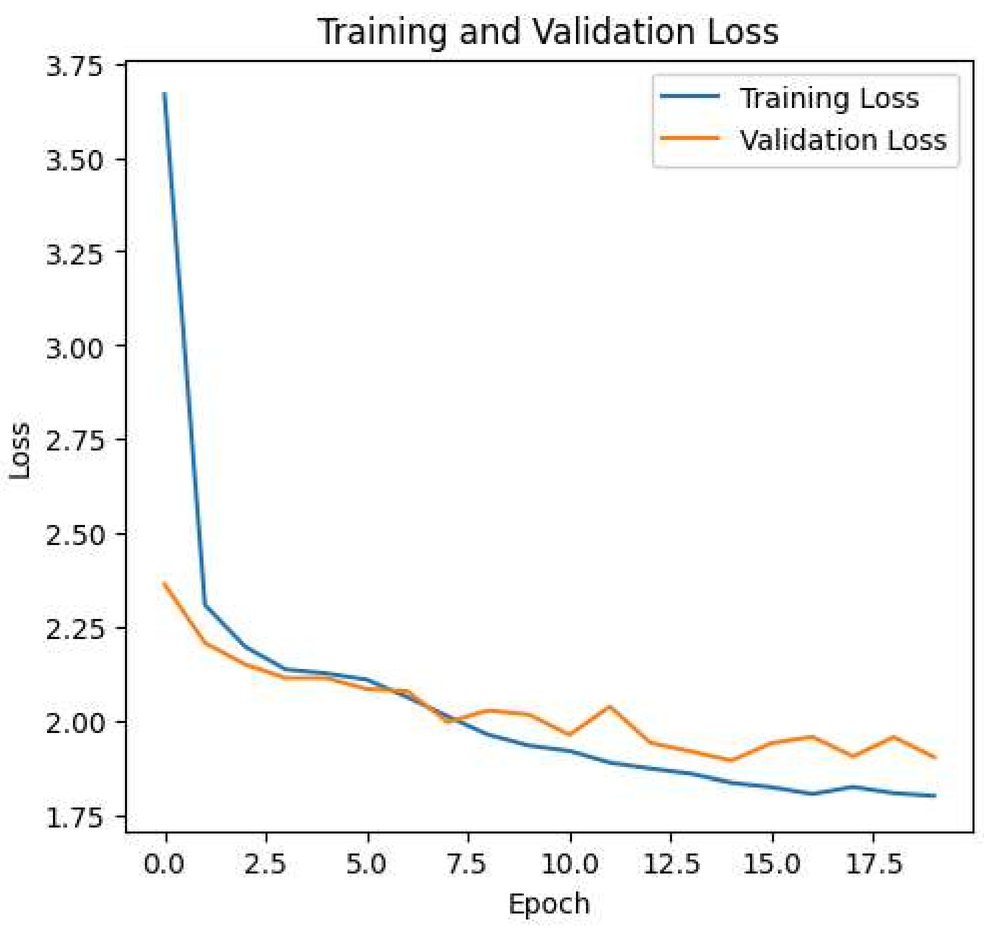
2. Applying Regularization

```
from tensorflow.keras import layers, models, regularizers

# Define the initial model
def create_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        # Add L1 and L2 regularization to the Dense layer
        layers.Dense(10, activation='softmax',
                     kernel_regularizer=regularizers.l1_l2(l1=0.01, l2=0.01))
    ])
    return model

# Create the model with L1 and L2 regularization
model_l1_l2 = create_model()

# Compile the model
model_l1_l2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history_l1_l2 = model_l1_l2.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))
```

```
Epoch 1/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - accuracy: 0.1224 - loss: 5.2392 - val_accuracy: 0.2100 - val_loss: 2.3653
Epoch 2/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.1471 - loss: 2.3400 - val_accuracy: 0.2600 - val_loss: 2.2091
Epoch 3/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.2367 - loss: 2.2172 - val_accuracy: 0.3020 - val_loss: 2.1517
Epoch 4/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.2705 - loss: 2.1424 - val_accuracy: 0.3020 - val_loss: 2.1157
Epoch 5/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 54ms/step - accuracy: 0.2908 - loss: 2.0978 - val_accuracy: 0.3020 - val_loss: 2.1156
Epoch 6/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.2815 - loss: 2.1099 - val_accuracy: 0.3020 - val_loss: 2.0865
Epoch 7/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 45ms/step - accuracy: 0.3277 - loss: 2.0592 - val_accuracy: 0.3020 - val_loss: 2.0811
Epoch 8/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.3368 - loss: 2.0260 - val_accuracy: 0.3640 - val_loss: 1.9979
Epoch 9/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.3673 - loss: 1.9762 - val_accuracy: 0.3060 - val_loss: 2.0293
Epoch 10/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 46ms/step - accuracy: 0.3954 - loss: 1.9056 - val_accuracy: 0.3520 - val_loss: 2.0182
Epoch 11/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 39ms/step - accuracy: 0.4028 - loss: 1.9104 - val_accuracy: 0.3740 - val_loss: 1.9645
Epoch 12/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.3858 - loss: 1.8776 - val_accuracy: 0.3540 - val_loss: 2.0398
Epoch 13/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.3951 - loss: 1.8682 - val_accuracy: 0.3820 - val_loss: 1.9431
Epoch 14/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.4002 - loss: 1.8572 - val_accuracy: 0.3720 - val_loss: 1.9201
Epoch 15/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 38ms/step - accuracy: 0.4308 - loss: 1.8308 - val_accuracy: 0.3760 - val_loss: 1.8961
Epoch 16/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 49ms/step - accuracy: 0.4252 - loss: 1.8209 - val_accuracy: 0.3700 - val_loss: 1.9424
Epoch 17/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.4193 - loss: 1.8237 - val_accuracy: 0.3660 - val_loss: 1.9592
Epoch 18/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.4077 - loss: 1.8501 - val_accuracy: 0.3740 - val_loss: 1.9067
Epoch 19/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.4381 - loss: 1.7764 - val_accuracy: 0.3640 - val_loss: 1.9585
Epoch 20/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.4249 - loss: 1.7824 - val_accuracy: 0.3920 - val_loss: 1.9051
```

```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history_l1_l2.history['accuracy'], label='Training Accuracy')
plt.plot(history_l1_l2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history_l1_l2.history['loss'], label='Training Loss')
plt.plot(history_l1_l2.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```





Training and Validation Accuracy:

The training accuracy shows a steady increase, indicating that the model is learning the training data well.

The validation accuracy also shows an upward trend, which is a good sign. This suggests that the model is now generalizing better to unseen data compared to before.

Training and Validation Loss:

The training loss decreases steadily, which means the model is fitting the training data well.

The validation loss also decreases, but not as smoothly as the training loss. This indicates that there is still some fluctuation, but overall, the model's performance on unseen data has improved.

Inference: The application of L1 and L2 regularization has helped in reducing overfitting. The model's performance on both training and validation data has improved, as seen by the steady increase in validation accuracy and the decrease in validation loss. This indicates that the regularization techniques have encouraged the model to maintain simplicity and avoid fitting too closely to the training data, thus improving generalization.

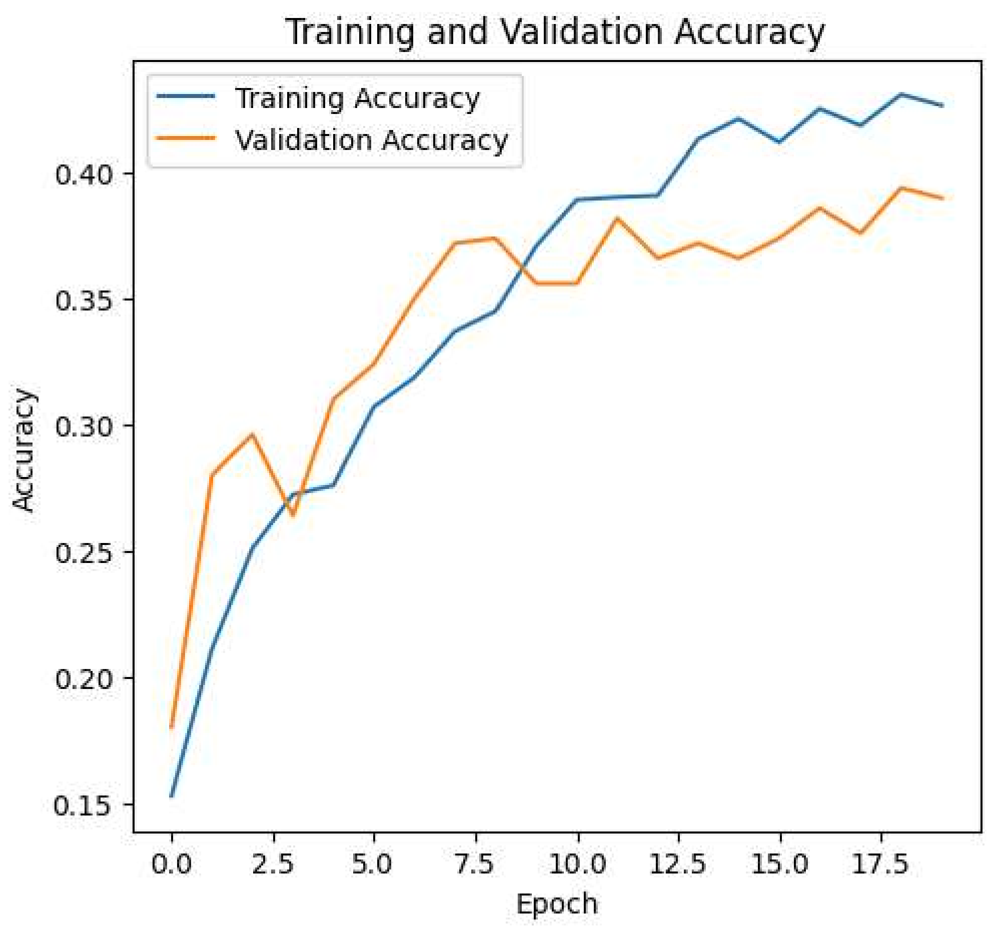
3. Early stopping

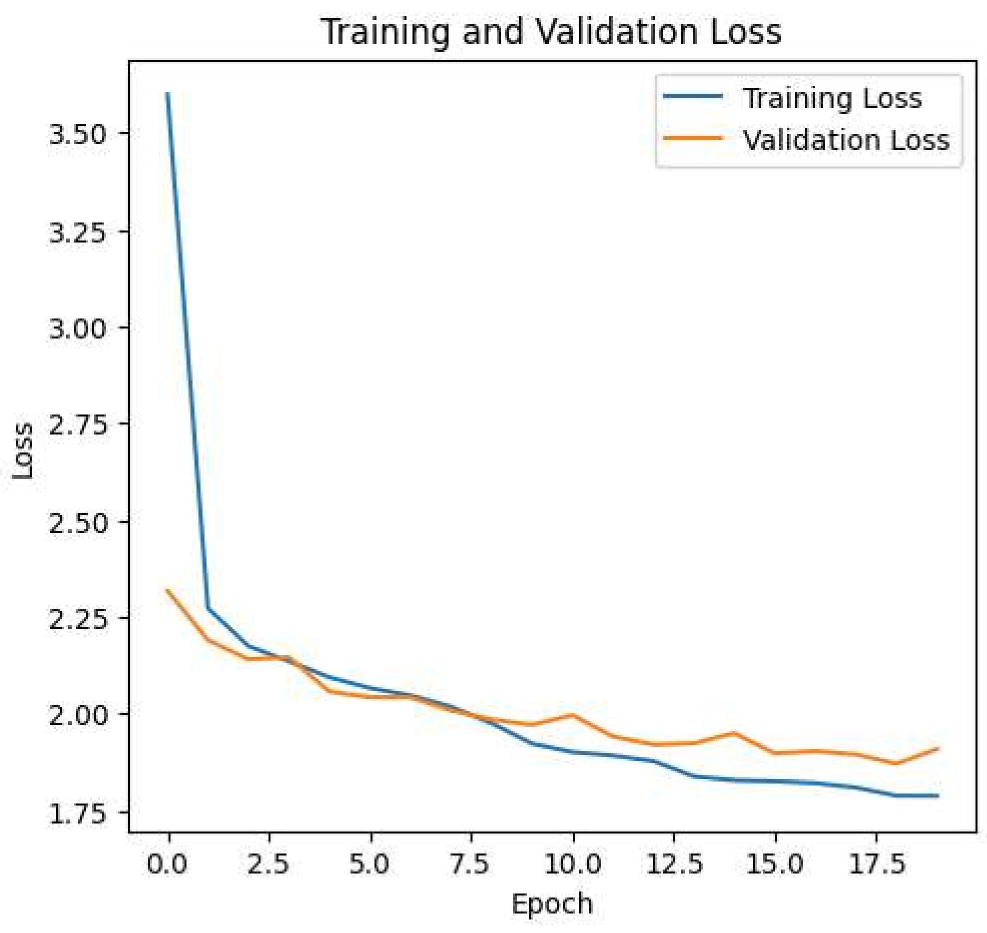
```
from tensorflow.keras.callbacks import EarlyStopping

# Define the EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=3,          # Number of epochs with no improvement after which training will stop
    restore_best_weights=True  # Restore the best weights when training stops
)

# Create the model with L1 and L2 regularization
model_l1_l2 = create_model()

# Compile the model
model_l1_l2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model with early stopping
history_l1_l2 = model_l1_l2.fit(
    x_train, y_train,
    epochs=20,  # Maximum number of epochs
    validation_data=(x_test, y_test),
    callbacks=[early_stopping]  # Add the EarlyStopping callback
)

# Evaluate the model
test_loss, test_accuracy = model_l1_l2.evaluate(x_test, y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Epoch 1/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 42ms/step - accuracy: 0.1255 - loss: 5.1718 - val_accuracy: 0.1800 - val_loss: 2.3181
Epoch 2/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 40ms/step - accuracy: 0.1989 - loss: 2.2928 - val_accuracy: 0.2800 - val_loss: 2.1903
```

```
Epoch 3/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.2444 - loss: 2.1863 - val_accuracy: 0.2960 - val_loss: 2.1410
Epoch 4/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.2739 - loss: 2.1307 - val_accuracy: 0.2640 - val_loss: 2.1461
Epoch 5/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.2651 - loss: 2.1299 - val_accuracy: 0.3100 - val_loss: 2.0572
Epoch 6/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 7s 59ms/step - accuracy: 0.2975 - loss: 2.0769 - val_accuracy: 0.3240 - val_loss: 2.0428
Epoch 7/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.3202 - loss: 2.0634 - val_accuracy: 0.3500 - val_loss: 2.0427
Epoch 8/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 41ms/step - accuracy: 0.3331 - loss: 2.0366 - val_accuracy: 0.3720 - val_loss: 2.0089
Epoch 9/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.3292 - loss: 1.9859 - val_accuracy: 0.3740 - val_loss: 1.9853
Epoch 10/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.3647 - loss: 1.9505 - val_accuracy: 0.3560 - val_loss: 1.9715
Epoch 11/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.3976 - loss: 1.9060 - val_accuracy: 0.3560 - val_loss: 1.9962
Epoch 12/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.3919 - loss: 1.8719 - val_accuracy: 0.3820 - val_loss: 1.9412
Epoch 13/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.3916 - loss: 1.8635 - val_accuracy: 0.3660 - val_loss: 1.9203
Epoch 14/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 39ms/step - accuracy: 0.4191 - loss: 1.8258 - val_accuracy: 0.3720 - val_loss: 1.9235
Epoch 15/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.4449 - loss: 1.8079 - val_accuracy: 0.3660 - val_loss: 1.9498
Epoch 16/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 40ms/step - accuracy: 0.4178 - loss: 1.8085 - val_accuracy: 0.3740 - val_loss: 1.8977
Epoch 17/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.4363 - loss: 1.8159 - val_accuracy: 0.3860 - val_loss: 1.9033
Epoch 18/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 50ms/step - accuracy: 0.4160 - loss: 1.8246 - val_accuracy: 0.3760 - val_loss: 1.8947
Epoch 19/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.4400 - loss: 1.7702 - val_accuracy: 0.3940 - val_loss: 1.8710
Epoch 20/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.4263 - loss: 1.7875 - val_accuracy: 0.3900 - val_loss: 1.9087
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.4156 - loss: 1.8288
Test Loss: 1.8709510564804077
Test Accuracy: 0.39399999380111694
```

```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history_l1_l2.history['accuracy'], label='Training Accuracy')
plt.plot(history_l1_l2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history_l1_l2.history['loss'], label='Training Loss')
plt.plot(history_l1_l2.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```





1. **Training and Validation Accuracy**:
   - Both the training and validation accuracy show an upward trend.
   - The training accuracy increases steadily throughout the epochs.
   - The validation accuracy peaks around epoch 10 and then slightly decreases, suggesting that the model's ability to generalize to new data improves up to a certain point but then starts to decline slightly.
2. **Training and Validation Loss**:
   - The training loss decreases steadily, indicating that the model fits the training data well.
   - The validation loss also decreases but shows more fluctuation compared to the training loss.

Inference: Early stopping has helped mitigate overfitting by halting the training process before the model starts to overfit the training data. The improvement in validation accuracy and the reduction in validation loss suggest better generalization to new data. However, the slight decrease in validation accuracy after epoch 10 indicates that the model might still need some fine-tuning or additional regularization techniques to further enhance its performance.

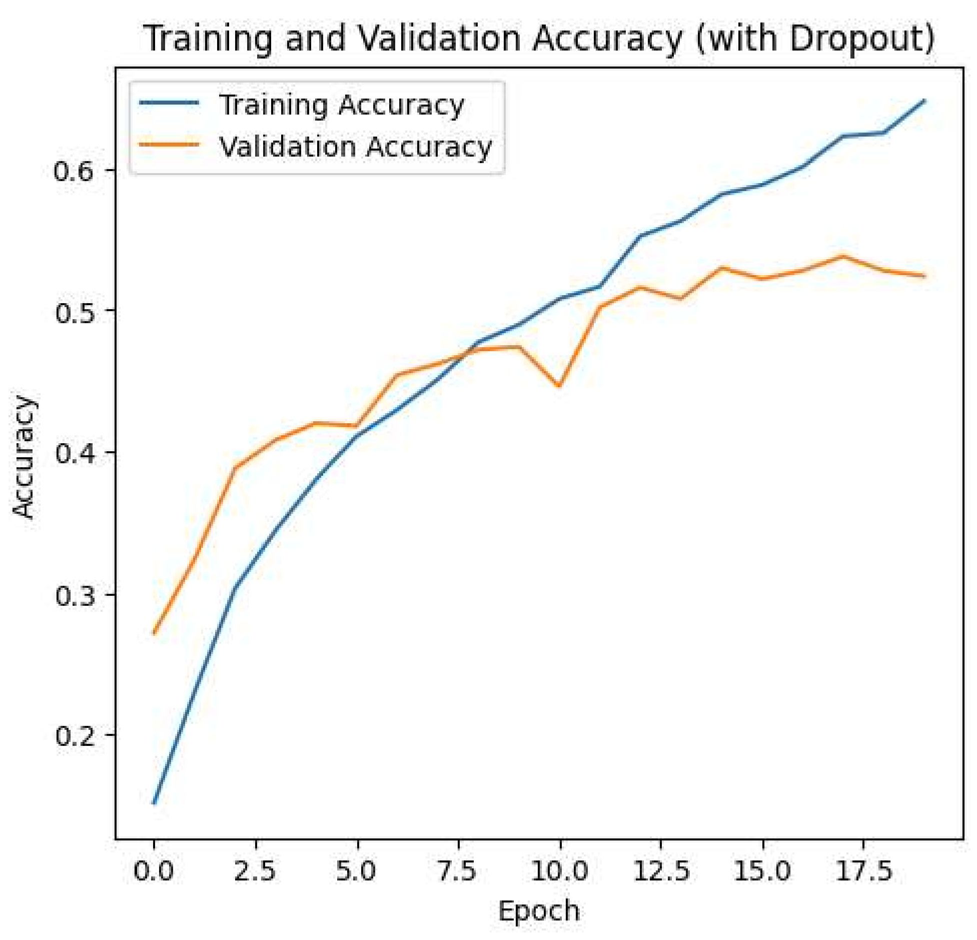
4.dropout

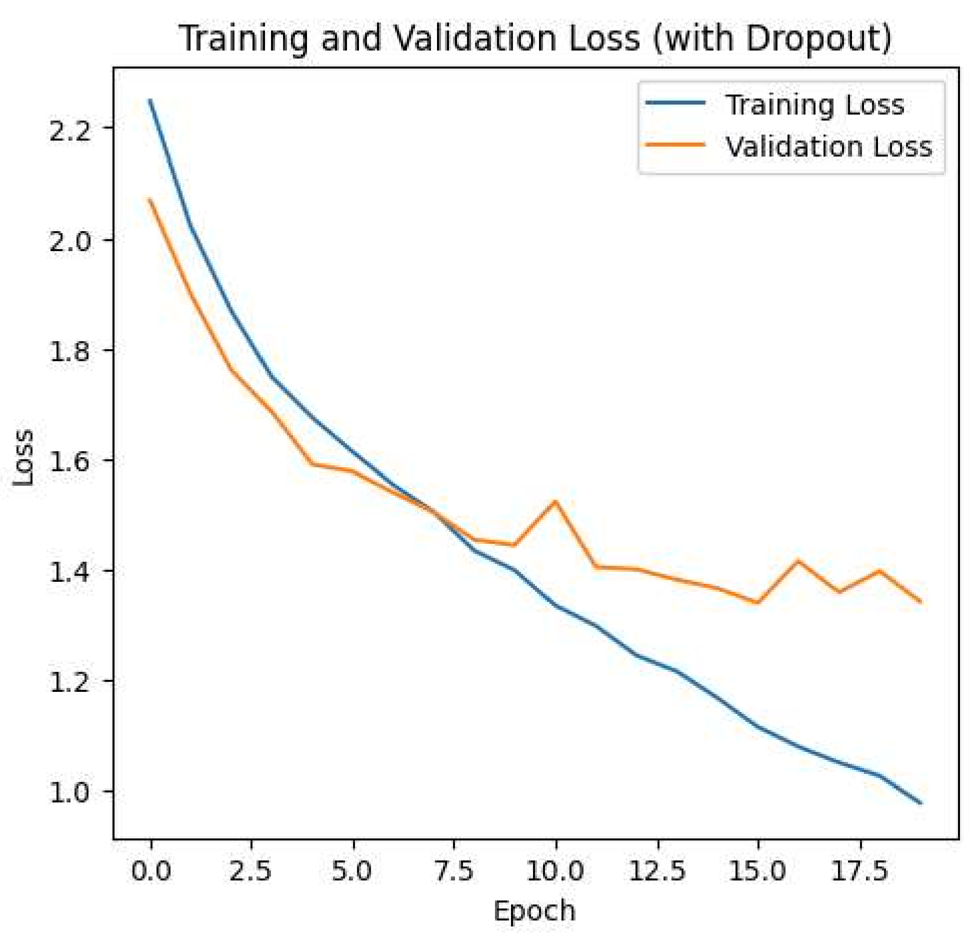
```python
from tensorflow.keras import layers, models, regularizers

# Define the model with dropout
def create_model_with_dropout():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),  # Dropout layer after the first pooling layer
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),  # Dropout layer after the second pooling layer
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),  # Dropout layer before the final Dense layer
        layers.Dense(10, activation='softmax')
    ])
    return model

# Create the model with dropout
model_dropout = create_model_with_dropout()

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

# Train the model
history_dropout = model_dropout.fit(
    x_train, y_train,
    epochs=20,
    validation_data=(x_test, y_test)
)

# Evaluate the model
```

```
test_loss, test_accuracy = model_dropout.evaluate(x_test, y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Epoch 1/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.1123 - loss: 2.3155 - val_accuracy: 0.2720 - val_loss: 2.0685
Epoch 2/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.2160 - loss: 2.0658 - val_accuracy: 0.3240 - val_loss: 1.8994
Epoch 3/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.2954 - loss: 1.8864 - val_accuracy: 0.3880 - val_loss: 1.7618
Epoch 4/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.3444 - loss: 1.7455 - val_accuracy: 0.4080 - val_loss: 1.6850
Epoch 5/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.3896 - loss: 1.6808 - val_accuracy: 0.4200 - val_loss: 1.5896
Epoch 6/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - accuracy: 0.4175 - loss: 1.6055 - val_accuracy: 0.4180 - val_loss: 1.5768
Epoch 7/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - accuracy: 0.4315 - loss: 1.5544 - val_accuracy: 0.4540 - val_loss: 1.5393
Epoch 8/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.4465 - loss: 1.5006 - val_accuracy: 0.4620 - val_loss: 1.5036
Epoch 9/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.4711 - loss: 1.4378 - val_accuracy: 0.4720 - val_loss: 1.4536
Epoch 10/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - accuracy: 0.5081 - loss: 1.3738 - val_accuracy: 0.4740 - val_loss: 1.4438
Epoch 11/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.5120 - loss: 1.3320 - val_accuracy: 0.4460 - val_loss: 1.5229
Epoch 12/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 10s 56ms/step - accuracy: 0.5272 - loss: 1.2881 - val_accuracy: 0.5020 - val_loss: 1.4040
Epoch 13/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 9s 44ms/step - accuracy: 0.5769 - loss: 1.2029 - val_accuracy: 0.5160 - val_loss: 1.3998
Epoch 14/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - accuracy: 0.5793 - loss: 1.1927 - val_accuracy: 0.5080 - val_loss: 1.3810
Epoch 15/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 10s 54ms/step - accuracy: 0.5843 - loss: 1.1741 - val_accuracy: 0.5300 - val_loss: 1.3655
Epoch 16/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 9s 45ms/step - accuracy: 0.5950 - loss: 1.0928 - val_accuracy: 0.5220 - val_loss: 1.3388
Epoch 17/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - accuracy: 0.5926 - loss: 1.0900 - val_accuracy: 0.5280 - val_loss: 1.4149
Epoch 18/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 10s 56ms/step - accuracy: 0.6036 - loss: 1.0636 - val_accuracy: 0.5380 - val_loss: 1.3584
Epoch 19/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 9s 45ms/step - accuracy: 0.6335 - loss: 1.0112 - val_accuracy: 0.5280 - val_loss: 1.3964
Epoch 20/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - accuracy: 0.6363 - loss: 0.9749 - val_accuracy: 0.5240 - val_loss: 1.3417
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5720 - loss: 1.3131
Test Loss: 1.3416987657546997
Test Accuracy: 0.5239999890327454
```

```
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history_dropout.history['accuracy'], label='Training Accuracy')
plt.plot(history_dropout.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy (with Dropout)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history_dropout.history['loss'], label='Training Loss')
plt.plot(history_dropout.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss (with Dropout)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```





Inference: The application of dropout has had a positive effect, improving the model's generalization ability. The increase in validation accuracy from 0.39 to 0.52 and the decrease in validation loss indicate that dropout helps prevent overfitting, leading to better performance on new data.

5.Data Augmentation

```
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the model (same as before)
def create_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    return model

# Create the model
model_aug = create_model()

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

# Define data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,      # Randomly rotate images by 15 degrees
    width_shift_range=0.1,  # Randomly shift images horizontally by 10%
    height_shift_range=0.1, # Randomly shift images vertically by 10%
    horizontal_flip=True,   # Randomly flip images horizontally
    zoom_range=0.1          # Randomly zoom images by 10%
)

# Fit the data augmentation to the training data
datagen.fit(x_train)

# Train the model with data augmentation
history_aug = model_aug.fit(
    datagen.flow(x_train, y_train, batch_size=64),  # Use augmented data
    epochs=20,
    validation_data=(x_test, y_test)
)

# Evaluate the model
test_loss, test_accuracy = model_aug.evaluate(x_test, y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call
rn_if_super_not_called()
━━━━━━━━━━━━━━━━ 8s 114ms/step - accuracy: 0.1686 - loss: 2.2391 - val_accuracy: 0.2160 - val_loss: 2.1456

━━━━━━━━━━━━━━━━ 5s 107ms/step - accuracy: 0.2899 - loss: 1.9508 - val_accuracy: 0.3380 - val_loss: 1.8363

━━━━━━━━━━━━━━━━ 6s 130ms/step - accuracy: 0.3650 - loss: 1.7543 - val_accuracy: 0.3860 - val_loss: 1.6739

━━━━━━━━━━━━━━━━ 5s 106ms/step - accuracy: 0.3898 - loss: 1.6685 - val_accuracy: 0.3980 - val_loss: 1.6527

━━━━━━━━━━━━━━━━ 6s 131ms/step - accuracy: 0.4134 - loss: 1.6280 - val_accuracy: 0.4000 - val_loss: 1.6279

━━━━━━━━━━━━━━━━ 5s 107ms/step - accuracy: 0.4357 - loss: 1.5339 - val_accuracy: 0.4500 - val_loss: 1.5608

━━━━━━━━━━━━━━━━ 6s 121ms/step - accuracy: 0.4328 - loss: 1.5233 - val_accuracy: 0.4180 - val_loss: 1.5343

━━━━━━━━━━━━━━━━ 5s 113ms/step - accuracy: 0.4879 - loss: 1.4641 - val_accuracy: 0.4280 - val_loss: 1.5690

━━━━━━━━━━━━━━━━ 5s 103ms/step - accuracy: 0.4770 - loss: 1.4616 - val_accuracy: 0.4360 - val_loss: 1.4818
0
━━━━━━━━━━━━━━━━ 6s 133ms/step - accuracy: 0.4876 - loss: 1.4057 - val_accuracy: 0.4760 - val_loss: 1.4460
0
━━━━━━━━━━━━━━━━ 5s 106ms/step - accuracy: 0.5013 - loss: 1.3654 - val_accuracy: 0.4540 - val_loss: 1.5364
0
━━━━━━━━━━━━━━━━ 6s 134ms/step - accuracy: 0.5420 - loss: 1.3299 - val_accuracy: 0.5120 - val_loss: 1.3773
0
━━━━━━━━━━━━━━━━ 5s 104ms/step - accuracy: 0.5375 - loss: 1.2982 - val_accuracy: 0.4880 - val_loss: 1.4091
0
━━━━━━━━━━━━━━━━ 5s 107ms/step - accuracy: 0.5549 - loss: 1.2577 - val_accuracy: 0.5260 - val_loss: 1.3666
0
━━━━━━━━━━━━━━━━ 6s 129ms/step - accuracy: 0.5454 - loss: 1.2462 - val_accuracy: 0.5080 - val_loss: 1.4046
0
━━━━━━━━━━━━━━━━ 5s 103ms/step - accuracy: 0.5627 - loss: 1.2475 - val_accuracy: 0.5220 - val_loss: 1.3665
0
━━━━━━━━━━━━━━━━ 7s 135ms/step - accuracy: 0.5807 - loss: 1.2039 - val_accuracy: 0.5340 - val_loss: 1.3542
0
━━━━━━━━━━━━━━━━ 5s 108ms/step - accuracy: 0.5741 - loss: 1.1967 - val_accuracy: 0.5360 - val_loss: 1.3872
0
━━━━━━━━━━━━━━━━ 6s 124ms/step - accuracy: 0.5473 - loss: 1.2416 - val_accuracy: 0.5380 - val_loss: 1.3713
0
━━━━━━━━━━━━━━━━ 9s 106ms/step - accuracy: 0.6027 - loss: 1.1579 - val_accuracy: 0.5520 - val_loss: 1.3303
━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.5826 - loss: 1.2824
 1.3302639722824097
acy: 0.5519999861717224
```

```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history_aug.history['accuracy'], label='Training Accuracy')
plt.plot(history_aug.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy (with Data Augmentation)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history_aug.history['loss'], label='Training Loss')
plt.plot(history_aug.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss (with Data Augmentation)')
```