Data Preparation:

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

(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)


datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)


print("Training images:", x_train.shape)
print("Training labels:", y_train.shape)
print("Testing images:", x_test.shape)
print("Testing labels:", y_test.shape)
```

```
Training images: (50000, 32, 32, 3)
Training labels: (50000, 10)
Testing images: (10000, 32, 32, 3)
Testing labels: (10000, 10)
```

Model Implementation:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

model = Sequential()
model.add(Flatten(input_shape=(32, 32, 3)))
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential m
  super().__init__(**kwargs)
```

```
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 512)	1,573,376
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 10)	2,570

Total params: 1,707,274 (6.51 MB)
Trainable params: 1,707,274 (6.51 MB)
Non-trainable params: 0 (0.00 B)

```
history = model.fit(
    x_train, y_train,
    epochs=10,
    batch_size=64,
    validation_data=(x_test, y_test)
)
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.3702 - loss: 1.7476 - val_accuracy: 0.4082 - val_loss: 1.6455
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.4172 - loss: 1.6300 - val_accuracy: 0.4403 - val_loss: 1.5715
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.4494 - loss: 1.5473 - val_accuracy: 0.4505 - val_loss: 1.5418
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 32ms/step - accuracy: 0.4596 - loss: 1.5140 - val_accuracy: 0.4547 - val_loss: 1.5387
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 32ms/step - accuracy: 0.4728 - loss: 1.4758 - val_accuracy: 0.4517 - val_loss: 1.5368
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 32ms/step - accuracy: 0.4867 - loss: 1.4427 - val_accuracy: 0.4662 - val_loss: 1.4770
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.4912 - loss: 1.4270 - val_accuracy: 0.4745 - val_loss: 1.4926
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 26s 33ms/step - accuracy: 0.5032 - loss: 1.3886 - val_accuracy: 0.4727 - val_loss: 1.4889
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.5115 - loss: 1.3674 - val_accuracy: 0.4659 - val_loss: 1.5024
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.5162 - loss: 1.3455 - val_accuracy: 0.4888 - val_loss: 1.4330
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.4889 - loss: 1.4310
Test Accuracy: 0.4887999892234802
```

Model Training:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
import pandas as pd

model = Sequential([
    Flatten(input_shape=(32, 32, 3)),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

learning_rate = 0.001
epochs = 20
batch_size = 64

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

history = model.fit(
    x_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(x_test, y_test)
)

history_df = pd.DataFrame(history.history)
history_df.to_csv("ann_training_history.csv", index=False)

model.save("ann_cifar10_model.h5")
```

```
Epoch 1/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 22ms/step - accuracy: 0.2708 - loss: 2.0211 - val_accuracy: 0.3771 - val_loss: 1.7381
Epoch 2/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.3905 - loss: 1.6956 - val_accuracy: 0.4261 - val_loss: 1.6362
Epoch 3/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.4284 - loss: 1.5991 - val_accuracy: 0.4447 - val_loss: 1.5607
Epoch 4/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.4442 - loss: 1.5506 - val_accuracy: 0.4583 - val_loss: 1.5312
Epoch 5/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 18ms/step - accuracy: 0.4619 - loss: 1.5015 - val_accuracy: 0.4397 - val_loss: 1.5656
Epoch 6/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.4709 - loss: 1.4822 - val_accuracy: 0.4671 - val_loss: 1.5170
Epoch 7/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.4844 - loss: 1.4443 - val_accuracy: 0.4502 - val_loss: 1.5655
Epoch 8/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 20ms/step - accuracy: 0.4858 - loss: 1.4331 - val_accuracy: 0.4702 - val_loss: 1.5051
Epoch 9/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 20ms/step - accuracy: 0.4947 - loss: 1.4118 - val_accuracy: 0.4770 - val_loss: 1.4862
Epoch 10/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 19ms/step - accuracy: 0.5067 - loss: 1.3794 - val_accuracy: 0.4815 - val_loss: 1.4574
Epoch 11/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.5115 - loss: 1.3691 - val_accuracy: 0.4733 - val_loss: 1.4949
Epoch 12/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.5126 - loss: 1.3614 - val_accuracy: 0.4929 - val_loss: 1.4311
Epoch 13/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.5251 - loss: 1.3294 - val_accuracy: 0.4903 - val_loss: 1.4448
Epoch 14/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.5283 - loss: 1.3220 - val_accuracy: 0.4929 - val_loss: 1.4253
Epoch 15/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.5324 - loss: 1.3106 - val_accuracy: 0.4917 - val_loss: 1.4286
Epoch 16/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.5399 - loss: 1.2910 - val_accuracy: 0.5030 - val_loss: 1.4090
Epoch 17/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.5396 - loss: 1.2890 - val_accuracy: 0.4914 - val_loss: 1.4351
Epoch 18/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.5466 - loss: 1.2675 - val_accuracy: 0.5033 - val_loss: 1.4050
Epoch 19/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.5553 - loss: 1.2508 - val_accuracy: 0.4974 - val_loss: 1.4146
Epoch 20/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 17ms/step - accuracy: 0.5513 - loss: 1.2544 - val_accuracy: 0.5071 - val_loss: 1.4080
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the
```

Model Evaluation:

```
import numpy as np
from sklearn.metrics import confusion_matrix

y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

cm = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:\n", cm)
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step
Confusion Matrix:
 [[534  19 116  15  49  21  30  42 125  49]
 [ 47 552  33  21   8  27  18  41  72 181]
 [ 47  13 420  56 111  89 135  90  21  18]
 [ 22  14 123 194  63 256 161  94  30  43]
 [ 34   3 141  27 407  53 161 138  22  14]
 [ 16   9  96 106  57 433 114 126  22  21]
 [  3   5  73  50  85  48 656  55   9  16]
 [ 26   2  57  25  79  58  38 658  12  45]
 [114  34  31  14  35  24  16  26 617  89]
 [ 44 125  19  25  17  21  39  68  42 600]]
Test Loss: 1.4080229997634888
Test Accuracy: 0.507099986076355
```

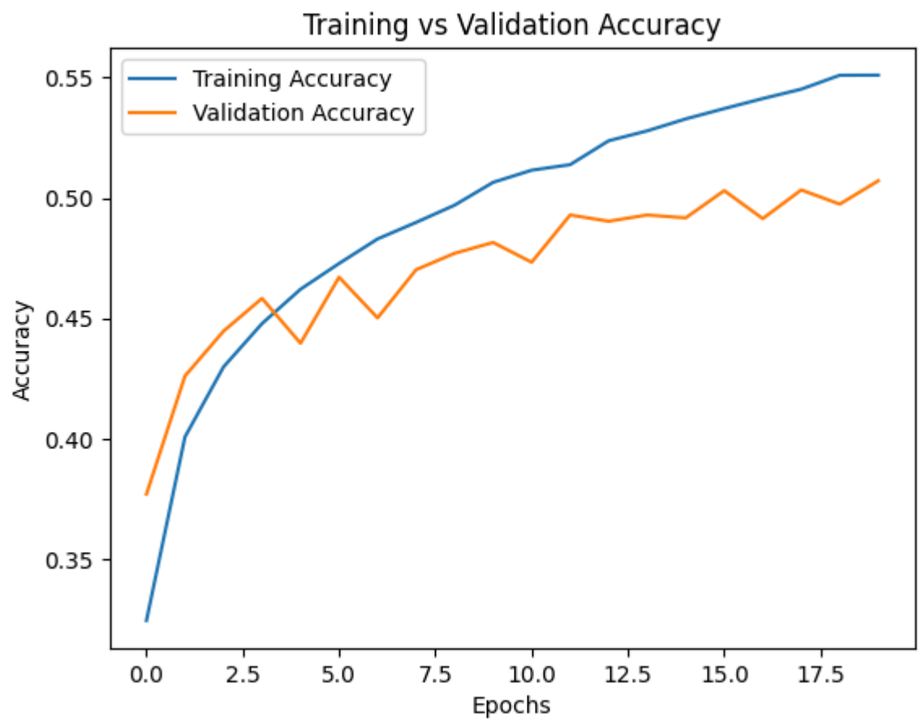
Performance Comparison:

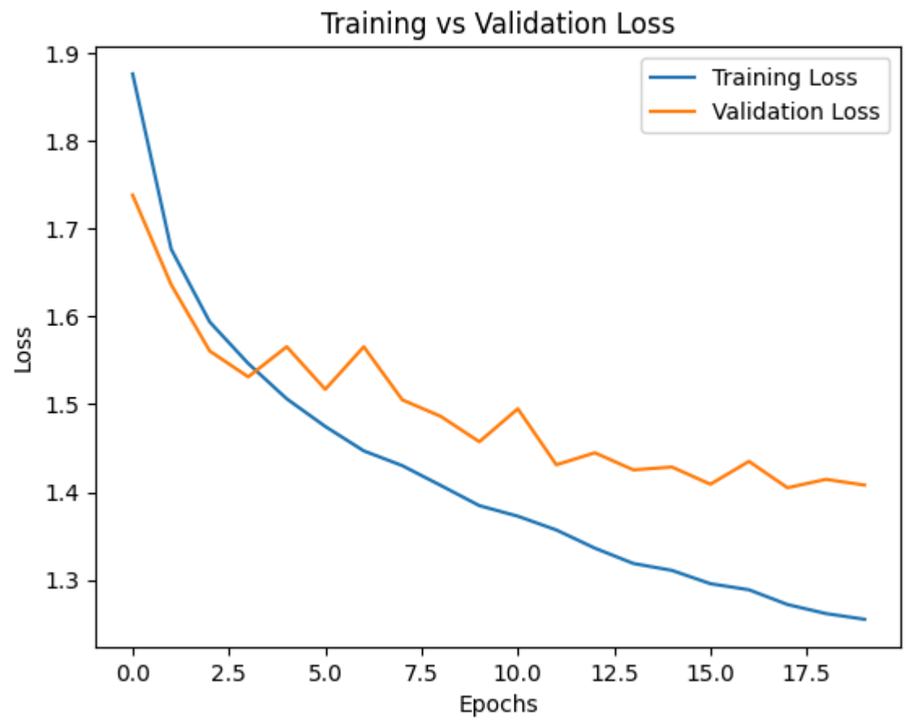
```
import matplotlib.pyplot as plt

plt.figure()
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()
plt.show()

plt.figure()
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()
```





Summary:

The CIFAR-10 dataset was loaded to perform image classification using an Artificial Neural Network (ANN). The dataset consists of 60,000 color images of size 32×32×3 belonging to 10 classes, with 50,000 images used for training and 10,000 images for testing. Data preprocessing involved normalizing pixel values to the range [0–1] and one-hot encoding the class labels. To improve generalization, data augmentation techniques such as rotation, width and height shifting, and horizontal flipping were applied.

An ANN model was developed using a Flatten layer followed by fully connected Dense layers with ReLU activation and a Softmax output layer. The model was trained using the Adam optimizer and categorical cross-entropy loss with a batch size of 64 for 10 and 20 epochs as shown in the experiments. The trained model achieved an approximate training accuracy of 55% and a testing accuracy of about 50%. The results indicate that model architecture, batch size, and number of epochs play a significant role in determining classification performance.