In this exam, you will use the CIFAR-10 dataset, a widely-used benchmark dataset in the field of computer vision.

Step 1: Load and Preprocess the CIFAR-10 Dataset & import needed libraries

```
import tensorflow as tf
cifar10 = tf.keras.datasets.cifar10
```

Step 2 : Data Preparation:

```
# Load CIFAR-10 data
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normalize the pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Convert class vectors to binary class matrices (one-hot encoding)
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

Step 3: Define the CNN Architecture and Compile the Model:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

Step 4: Train the Model

```
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 68s 43ms/step - loss: 1.7266 - accuracy: 0.3583 - val_loss: 1.3975
Epoch 2/10
1563/1563 [==============================] - 68s 44ms/step - loss: 1.3743 - accuracy: 0.5092 - val_loss: 1.1451
Epoch 3/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.2357 - accuracy: 0.5637 - val_loss: 1.0867
Epoch 4/10
1563/1563 [==============================] - 68s 44ms/step - loss: 1.1368 - accuracy: 0.5982 - val_loss: 1.0205
Epoch 5/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.0662 - accuracy: 0.6258 - val_loss: 0.9549
Epoch 6/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.0046 - accuracy: 0.6518 - val_loss: 0.9260
Epoch 7/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.9562 - accuracy: 0.6679 - val_loss: 0.9665
Epoch 8/10
1563/1563 [==============================] - 71s 45ms/step - loss: 0.9151 - accuracy: 0.6831 - val_loss: 0.8833
Epoch 9/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.8735 - accuracy: 0.6992 - val_loss: 0.8503
Epoch 10/10
```

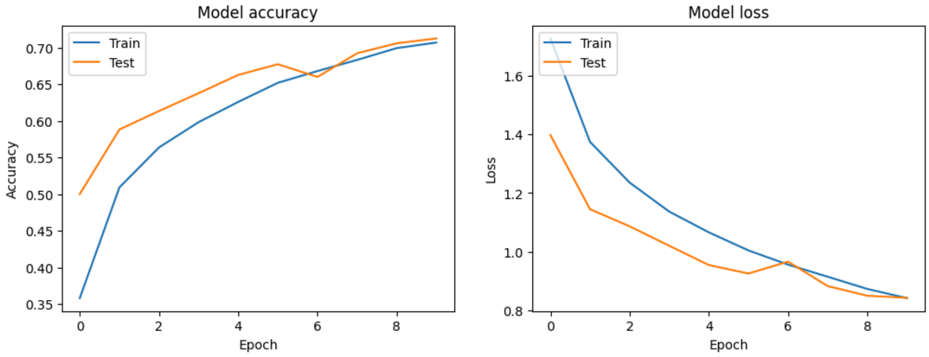
```
1563/1563 [==============================] - 69s 44ms/step - loss: 0.8423 - accuracy: 0.7067 - val_loss: 0.8433
```

Step 5: Model Evaluation:

- Evaluate the model: Use the test dataset to evaluate the model's accuracy and performance metrics.
- Plot results: Visualize training and validation accuracy and loss.

```
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nTest accuracy:', test_acc)

# Plot training & validation accuracy values
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
313/313 - 3s - loss: 0.8433 - accuracy: 0.7124 - 3s/epoch - 11ms/step

Test accuracy: 0.7124000191688538
```



Step 6: Model Improvement:

- Experimentation: Try different architectures, hyperparameters, and data augmentation techniques to enhance model performance.
- Regularization

```
# Experimenting with a different architecture and adding data augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create an image data generator
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
```

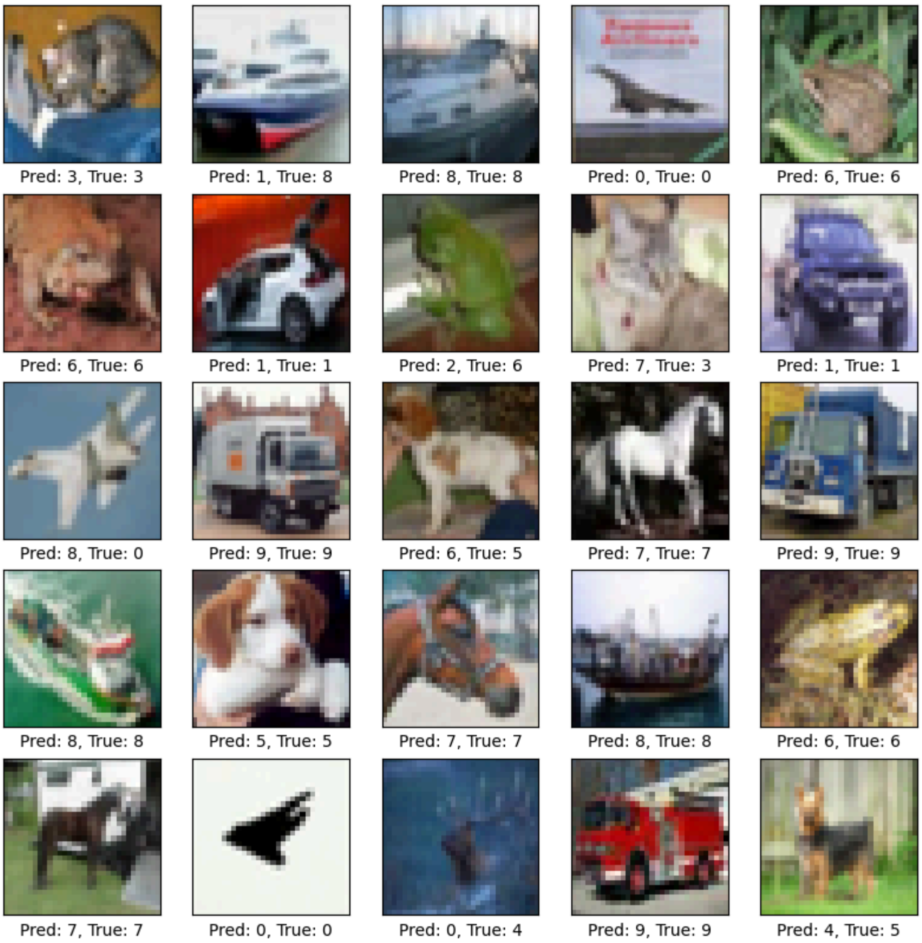
```
)
datagen.fit(train_images)

# Define a new model architecture
model_improved = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

# Train the improved model
history_improved = model_improved.fit(datagen.flow(train_images, train_labels, batch_size=64),
                                      epochs=10,
                                      validation_data=(test_images, test_labels))
```

```
Epoch 1/10
782/782 [==============================] - 104s 127ms/step - loss: 1.7971 - accuracy: 0.3332 - val_loss: 1.4744
Epoch 2/10
782/782 [==============================] - 95s 121ms/step - loss: 1.4764 - accuracy: 0.4689 - val_loss: 1.2767
Epoch 3/10
782/782 [==============================] - 95s 122ms/step - loss: 1.3561 - accuracy: 0.5167 - val_loss: 1.2747
Epoch 4/10
782/782 [==============================] - 96s 122ms/step - loss: 1.2730 - accuracy: 0.5490 - val_loss: 1.0681
Epoch 5/10
782/782 [==============================] - 97s 124ms/step - loss: 1.1982 - accuracy: 0.5774 - val_loss: 1.0281
Epoch 6/10
782/782 [==============================] - 93s 118ms/step - loss: 1.1534 - accuracy: 0.5951 - val_loss: 0.9773
Epoch 7/10
782/782 [==============================] - 95s 121ms/step - loss: 1.1066 - accuracy: 0.6142 - val_loss: 0.9778
Epoch 8/10
782/782 [==============================] - 94s 120ms/step - loss: 1.0742 - accuracy: 0.6260 - val_loss: 0.9964
Epoch 9/10
782/782 [==============================] - 93s 119ms/step - loss: 1.0396 - accuracy: 0.6380 - val_loss: 0.9771
Epoch 10/10
782/782 [==============================] - 95s 121ms/step - loss: 1.0163 - accuracy: 0.6479 - val_loss: 0.8979
```

Step 7: Make Predictions (optional):

```
# Make predictions on the test images
predictions = model_improved.predict(test_images)

# Show a few test images with their predicted labels
import numpy as np

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions[i])
    true_label = np.argmax(test_labels[i])
    plt.xlabel(f"Pred: {predicted_label}, True: {true_label}")
plt.show()
```

```
313/313 [==============================] - 5s 17ms/step
```



Double-click (or enter) to edit