```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import streamlit as st
from PIL import Image
import cv2
from sklearn.metrics import classification_report
from kerastuner.tuners import RandomSearch

# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalizing the data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Class names for CIFAR-10
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
               'Dog', 'Frog', 'Horse', 'Ship', 'Truck']
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Data Augmentation
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(X_train)
```

```
# Define CNN model architecture with dropout for regularization
def create_model(optimizer='adam', learning_rate=0.001):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.2))  # Dropout added for regularization
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(0.3))  # Dropout added for regularization
    model.add(layers.Dense(10, activation='softmax'))

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

# Hyperparameter tuning
def model_builder(hp):
    optimizer = hp.Choice('optimizer', ['adam', 'sgd'])
    learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-1,
sampling='LOG')
    model = create_model(optimizer=optimizer, learning_rate=learning_rate)
    return model

tuner = RandomSearch(
    model_builder,
    objective='val_accuracy',
```

```
    max_trials=10,
    executions_per_trial=1,
    directory='project_dir',
    project_name='cifar10_tuning'
)

# Train the model with data augmentation
tuner.search(datagen.flow(X_train, y_train, batch_size=64),
             epochs=10,
             validation_data=(X_test, y_test))

# Get the best model
best_model = tuner.get_best_models(num_models=1)[0]

# Evaluate the best model
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')

# Save the trained model's weights
best_model.save_weights('cifar10_model.weights.h5')

# Load model function for Streamlit
@st.cache_resource
def load_model():
    model = create_model()
    model.load_weights('cifar10_model.weights.h5')
    return model

# Load the trained model
model = load_model()
```

```
# Function to preprocess uploaded image
def preprocess_image(img):
    img = cv2.resize(np.array(img), (32, 32))
    img = np.expand_dims(img, axis=0)
    img = img / 255.0  # Normalize the image
    return img

# Streamlit interface
st.title('CIFAR-10 Image Classifier')
st.write("Upload an image to classify it!")

uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "png"])

if uploaded_file is not None:
    # Display the uploaded image
    image = Image.open(uploaded_file)
    st.image(image, caption='Uploaded Image.', use_column_width=True)

    # Preprocess the image
    processed_image = preprocess_image(image)

    # Make a prediction
    prediction = model.predict(processed_image)
    predicted_class = class_names[np.argmax(prediction)]

    # Display the prediction
    st.write(f"Prediction: *{predicted_class}*")

# Additional model evaluation (Precision, Recall, F1-Score)
```

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

# Generate classification report
report = classification_report(y_true, y_pred_classes, target_names=class_names)

# Print classification report (includes precision, recall, and F1-score)
print(report)
```