Aim: For an image classification challenge create and train a ConvNet in python using Tensorflow. Also try to improve the performance of the model by applying various hyperparameter tuning to reduce overfitting or underfitting problem that might occur. Maintain graphs of comparisons.

Software requirement for Python: Jupyter Notebook

Theory:

To implement an image classification challenge using Tensorflow, follow these steps:

1. Load the dataset: Load the image dataset you want to use for the classification task. You can use popular image datasets such as MNIST, CIFAR-10, or ImageNet.
2. Preprocess the dataset: Preprocess the image dataset by resizing the images, normalizing the pixel values, and dividing it into training, validation, and testing sets.
3. Define the ConvNet architecture: Define the architecture of the ConvNet using the Keras Sequential model. Add Conv2D layers, MaxPooling2D layers, and Dropout layers as required.
4. Compile the model: Compile the model using an optimizer, loss function, and evaluation metric.
5. Train the model: Train the model on the training set for a fixed number of epochs using the fit() method. Use the validation set to monitor the performance of the model and avoid overfitting.
6. Evaluate the model: Evaluate the performance of the model on the testing set using the evaluate() method.
7. Hyperparameter tuning: Use techniques such as grid search, random search, and Bayesian optimization to find the optimal hyperparameters for the model. Tune the learning rate, batch size, number of filters, and other hyperparameters to improve the performance of the model.
8. Regularization techniques: Apply regularization techniques such as L1/L2 regularization, dropout, and data augmentation to reduce overfitting or underfitting problems that might occur.

Procedure:

1. This code is implementing an image classification challenge using a Convolutional Neural Network (ConvNet) in Python with TensorFlow. The CIFAR-10 dataset is being used for this task, which consists of 50,000 training images and 10,000 test images, with 10 different classes of objects. The following steps are performed in the code:
2. Load and preprocess the CIFAR-10 dataset: The dataset is loaded into memory and preprocessed by scaling the pixel values of the images to the range of [0,1].
3. Define the ConvNet architecture: A sequential model is created using the Keras API with a series of convolutional layers, max pooling layers, dropout layers, and fully connected layers. The architecture consists of 2 blocks of convolutional layers with a max pooling layer and a dropout layer following each block, followed by a flatten layer, two dense layers, and a final output layer with softmax activation.
4. Compile the model: The model is compiled with the Adam optimizer and 'sparse_categorical_crossentropy' as the loss function.
5. Set up data augmentation: Data augmentation is used to increase the size of the dataset and reduce overfitting. The ImageDataGenerator class from Keras is used to apply random transformations to the training images, such as rotation, width and height shifts, and horizontal flipping.

6. Train the model: The fit method of the model is called with the data generator and other hyperparameters, such as batch size and number of epochs.
7. Evaluate the model: The evaluate method is called on the test set to get the test loss and accuracy of the model.
8. Plot the accuracy and loss curves: The training and validation accuracy and loss curves are plotted using the Matplotlib library to visualize the performance of the model over the training epochs.

Conclusion: Thus, we have studied an image classification challenge to create and train a ConvNet in python using Tensorflow.

Program Code and Output

```
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
from tensorflow.keras.preprocessing.image import ImageDataGenerator
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 / 255.0
x_test = x_test / 255.0

# Define the ConvNet architecture
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Dense(10, activation='softmax'))

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

# Set up data augmentation
datagen = ImageDataGenerator(rotation_range=15,
                  width_shift_range=0.1,
                  height_shift_range=0.1,
                  horizontal_flip=True)

# Train the model
history = model.fit(datagen.flow(x_train, y_train, batch_size=128),
            epochs=50,
            validation_data=(x_test, y_test))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_acc:.4f}')

# Plot the accuracy and loss curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

OUTPUT OF THE CODE

>>> %Run 'GROUP C_4.py'

Epoch 1/50

391/391 [==============================] - 99s 250ms/step - loss: 1.7248 - accuracy: 0.1030 - val_loss: 1.3772 - val_accuracy: 0.0655

Epoch 2/50

391/391 [==============================] - 127s 325ms/step - loss: 1.3879 - accuracy: 0.0958 - val_loss: 1.2349 - val_accuracy: 0.1099

Epoch 3/50

391/391 [==============================] - 114s 292ms/step - loss: 1.2416 - accuracy: 0.0983 - val_loss: 1.0503 - val_accuracy: 0.0805

Epoch 4/50

391/391 [==============================] - 116s 297ms/step - loss: 1.1595 - accuracy: 0.0993 - val_loss: 0.9859 - val_accuracy: 0.0880

Epoch 5/50

391/391 [==============================] - 118s 303ms/step - loss: 1.0877 - accuracy: 0.0986 - val_loss: 0.9002 - val_accuracy: 0.0991

Epoch 6/50

391/391 [==============================] - 119s 304ms/step - loss: 1.0244 - accuracy: 0.0999 - val_loss: 0.8383 - val_accuracy: 0.0931

Epoch 7/50

391/391 [==============================] - 131s 335ms/step - loss: 0.9825 - accuracy: 0.0997 - val_loss: 0.7868 - val_accuracy: 0.1026

Epoch 8/50

391/391 [==============================] - 141s 360ms/step - loss: 0.9517 - accuracy: 0.1001 - val_loss: 0.7742 - val_accuracy: 0.0811

Epoch 9/50

391/391 [==============================] - 147s 377ms/step - loss: 0.9256 - accuracy: 0.1006 - val_loss: 0.7405 - val_accuracy: 0.1036

Epoch 10/50

391/391 [==============================] - 142s 362ms/step - loss: 0.8991 - accuracy: 0.0996 - val_loss: 0.7711 - val_accuracy: 0.0972

Epoch 11/50

391/391 [==============================] - 139s 357ms/step - loss: 0.8799 - accuracy: 0.1024 - val_loss: 0.7556 - val_accuracy: 0.1119

Epoch 12/50

391/391 [==============================] - 135s 344ms/step - loss: 0.8639 - accuracy: 0.1008 - val_loss: 0.8286 - val_accuracy: 0.0987

Epoch 13/50

391/391 [==============================] - 82s 209ms/step - loss: 0.8410 - accuracy: 0.1019 - val_loss: 0.7132 - val_accuracy: 0.1033

Epoch 14/50

391/391 [==============================] - 87s 223ms/step - loss: 0.8264 - accuracy: 0.1009 - val_loss: 0.7772 - val_accuracy: 0.0989

Epoch 15/50

391/391 [==============================] - 97s 247ms/step - loss: 0.8172 - accuracy: 0.1035 - val_loss: 0.7462 - val_accuracy: 0.0923

Epoch 16/50

391/391 [==============================] - 91s 234ms/step - loss: 0.7979 - accuracy: 0.1023 - val_loss: 0.6487 - val_accuracy: 0.0955

Epoch 17/50

391/391 [==============================] - 94s 239ms/step - loss: 0.8003 - accuracy: 0.1029 - val_loss: 0.6650 - val_accuracy: 0.1051

Epoch 18/50

391/391 [==============================] - 90s 229ms/step - loss: 0.7865 - accuracy: 0.1022 - val_loss: 0.6724 - val_accuracy: 0.0995

Epoch 19/50

391/391 [==============================] - 88s 225ms/step - loss: 0.7717 - accuracy: 0.1033 - val_loss: 0.6833 - val_accuracy: 0.0951

Epoch 20/50

391/391 [==============================] - 88s 225ms/step - loss: 0.7625 - accuracy: 0.1020 - val_loss: 0.6469 - val_accuracy: 0.0969

Epoch 21/50

391/391 [==============================] - 87s 222ms/step - loss: 0.7555 - accuracy: 0.1020 - val_loss: 0.6600 - val_accuracy: 0.1045

Epoch 22/50

391/391 [==============================] - 87s 223ms/step - loss: 0.7479 - accuracy: 0.1013 - val_loss: 0.6610 - val_accuracy: 0.0972

Epoch 23/50

391/391 [==============================] - 90s 229ms/step - loss: 0.7477 - accuracy: 0.1023 - val_loss: 0.6384 - val_accuracy: 0.1121

Epoch 24/50

391/391 [==============================] - 91s 232ms/step - loss: 0.7366 - accuracy: 0.1028 - val_loss: 0.7233 - val_accuracy: 0.0957

Epoch 25/50

391/391 [==============================] - 88s 225ms/step - loss: 0.7281 - accuracy: 0.1023 - val_loss: 0.6643 - val_accuracy: 0.1054

Epoch 26/50

391/391 [==============================] - 89s 228ms/step - loss: 0.7222 - accuracy: 0.1020 - val_loss: 0.6386 - val_accuracy: 0.0993

Epoch 27/50

391/391 [==============================] - 88s 225ms/step - loss: 0.7097 - accuracy: 0.1032 - val_loss: 0.6283 - val_accuracy: 0.1012

Epoch 28/50

391/391 [==============================] - 88s 225ms/step - loss: 0.7086 - accuracy: 0.1021 - val_loss: 0.6461 - val_accuracy: 0.0954

Epoch 29/50

391/391 [==============================] - 90s 231ms/step - loss: 0.7081 - accuracy: 0.1023 - val_loss: 0.6026 - val_accuracy: 0.1013

Epoch 30/50

391/391 [==============================] - 89s 227ms/step - loss: 0.6984 - accuracy: 0.1027 - val_loss: 0.5945 - val_accuracy: 0.0939

Epoch 31/50

391/391 [==============================] - 88s 224ms/step - loss: 0.6926 - accuracy: 0.1016 - val_loss: 0.6842 - val_accuracy: 0.0996

Epoch 32/50

391/391 [==============================] - 88s 224ms/step - loss: 0.6868 - accuracy: 0.1026 - val_loss: 0.6042 - val_accuracy: 0.0948

Epoch 33/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6919 - accuracy: 0.1023 - val_loss: 0.6046 - val_accuracy: 0.0975

Epoch 34/50

391/391 [==============================] - 89s 227ms/step - loss: 0.6845 - accuracy: 0.1020 - val_loss: 0.6066 - val_accuracy: 0.0944

Epoch 35/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6814 - accuracy: 0.1030 - val_loss: 0.6341 - val_accuracy: 0.1030

Epoch 36/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6771 - accuracy: 0.1028 - val_loss: 0.6575 - val_accuracy: 0.0953

Epoch 37/50

391/391 [==============================] - 89s 227ms/step - loss: 0.6741 - accuracy: 0.1027 - val_loss: 0.5642 - val_accuracy: 0.0929

Epoch 38/50

391/391 [==============================] - 89s 227ms/step - loss: 0.6721 - accuracy: 0.1029 - val_loss: 0.6416 - val_accuracy: 0.0919

Epoch 39/50

391/391 [==============================] - 90s 231ms/step - loss: 0.6735 - accuracy: 0.1024 - val_loss: 0.5674 - val_accuracy: 0.1008

Epoch 40/50

391/391 [==============================] - 86s 220ms/step - loss: 0.6622 - accuracy: 0.1032 - val_loss: 0.6514 - val_accuracy: 0.0896

Epoch 41/50

391/391 [==============================] - 91s 233ms/step - loss: 0.6578 - accuracy: 0.1031 - val_loss: 0.6505 - val_accuracy: 0.1189

Epoch 42/50

391/391 [==============================] - 93s 237ms/step - loss: 0.6585 - accuracy: 0.1028 - val_loss: 0.5454 - val_accuracy: 0.1092

Epoch 43/50

391/391 [==============================] - 90s 231ms/step - loss: 0.6624 - accuracy: 0.1030 - val_loss: 0.6359 - val_accuracy: 0.1016

Epoch 44/50

391/391 [==============================] - 92s 234ms/step - loss: 0.6503 - accuracy: 0.1038 - val_loss: 0.5993 - val_accuracy: 0.0920

Epoch 45/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6499 - accuracy: 0.1038 - val_loss: 0.5999 - val_accuracy: 0.0940

Epoch 46/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6500 - accuracy: 0.1022 - val_loss: 0.6407 - val_accuracy: 0.0937

Epoch 47/50

391/391 [==============================] - 89s 226ms/step - loss: 0.6467 - accuracy: 0.1020 - val_loss: 0.6009 - val_accuracy: 0.1112

Epoch 48/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6494 - accuracy: 0.1030 - val_loss: 0.6091 - val_accuracy: 0.0937

Epoch 49/50

391/391 [==============================] - 88s 225ms/step - loss: 0.6468 - accuracy: 0.1035 - val_loss: 0.5794 - val_accuracy: 0.1005

Epoch 50/50

391/391 [==============================] - 88s 226ms/step - loss: 0.6400 - accuracy: 0.1018 - val_loss: 0.5958 - val_accuracy: 0.0968

313/313 - 4s - loss: 0.5958 - accuracy: 0.0968 - 4s/epoch - 12ms/step

Test Loss: 0.5958

Test Accuracy: 0.0968

Graphs of comparison for Accuracy Vs Epochs and Loss vs Epochs