Practical – 4

Practical: Hands-on training of convolutional neural networks.

Tasks: Preparing datasets for training CNNs. Implementing a CNN architecture using TensorFlow or Keras. Training the CNN model on a dataset for image classification or object detection task. Fine-tuning pre-trained CNN models for specific tasks.

Code:

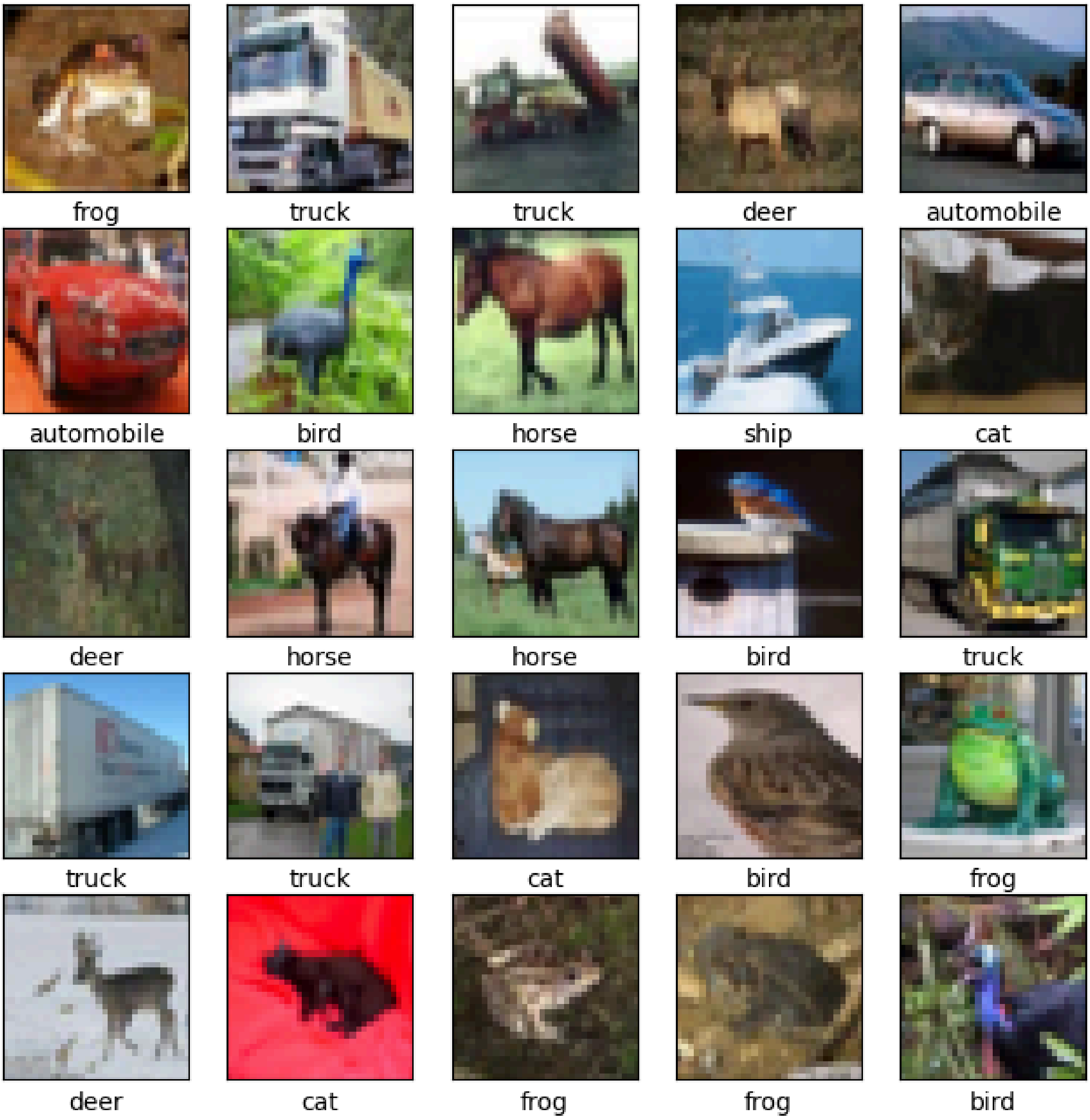
```
#Import Necessary Libraries:
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

#Load and Normalize the Dataset:
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

#Define Class Names:
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']


#Visualize the Dataset:
plt.figure(figsize=(8,8))
for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(train_images[i])
   plt.xlabel(class_names[train_labels[i][0]])
plt.show()

   # The CIFAR labels happen to be arrays, which is why we need the extra index
```



#Define the CNN Model:

```
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.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.Dense(10))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

#Compile the Model:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

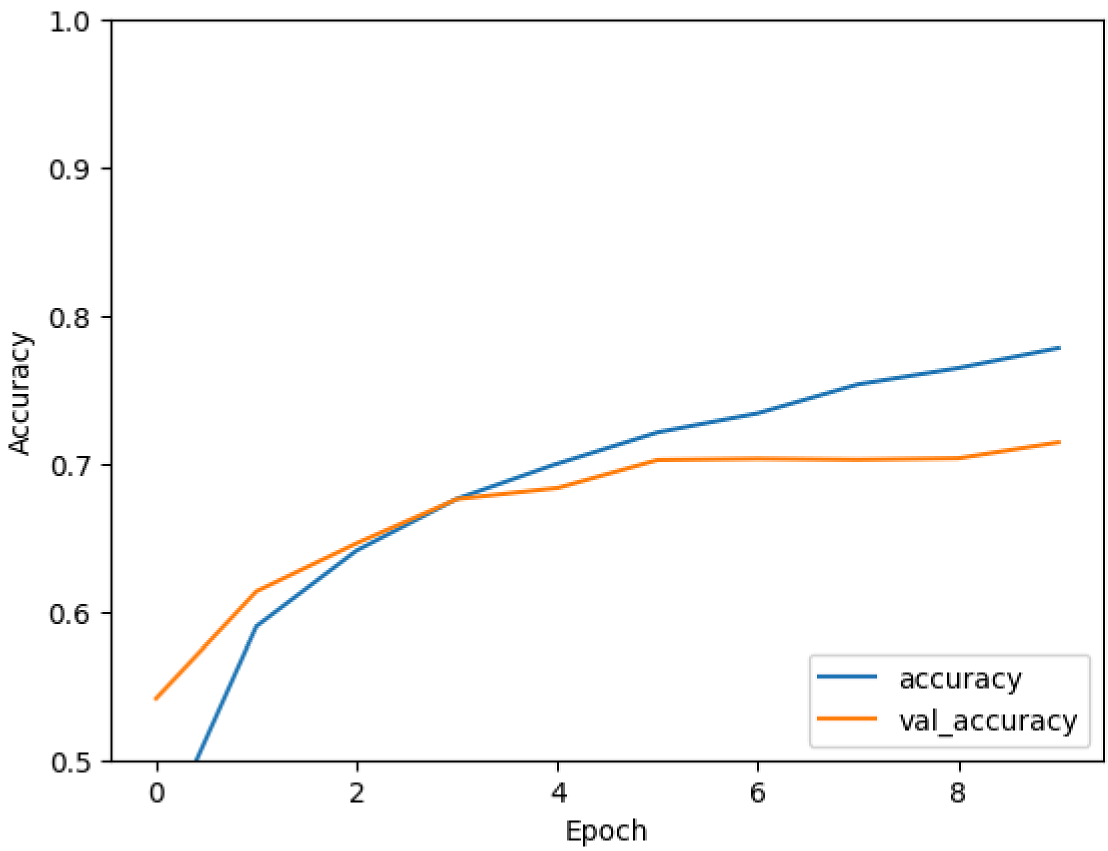
#Train the Model:

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

```
Epoch 1/10
1563/1563 [==============================] - 88s 55ms/step - loss: 1.5373 - accuracy: 0.4390 - val_loss: 1.2630 - val_accuracy: 0.5416
Epoch 2/10
1563/1563 [==============================] - 79s 50ms/step - loss: 1.1589 - accuracy: 0.5901 - val_loss: 1.0831 - val_accuracy: 0.6137
Epoch 3/10
1563/1563 [==============================] - 72s 46ms/step - loss: 1.0138 - accuracy: 0.6413 - val_loss: 1.0022 - val_accuracy: 0.6462
Epoch 4/10
1563/1563 [==============================] - 74s 47ms/step - loss: 0.9212 - accuracy: 0.6762 - val_loss: 0.9207 - val_accuracy: 0.6761
Epoch 5/10
1563/1563 [==============================] - 76s 48ms/step - loss: 0.8520 - accuracy: 0.7001 - val_loss: 0.9179 - val_accuracy: 0.6836
Epoch 6/10
1563/1563 [==============================] - 75s 48ms/step - loss: 0.7979 - accuracy: 0.7213 - val_loss: 0.8768 - val_accuracy: 0.7026
Epoch 7/10
1563/1563 [==============================] - 73s 47ms/step - loss: 0.7548 - accuracy: 0.7342 - val_loss: 0.8696 - val_accuracy: 0.7034
Epoch 8/10
1563/1563 [==============================] - 75s 48ms/step - loss: 0.7043 - accuracy: 0.7537 - val_loss: 0.8748 - val_accuracy: 0.7028
Epoch 9/10
1563/1563 [==============================] - 75s 48ms/step - loss: 0.6685 - accuracy: 0.7647 - val_loss: 0.8786 - val_accuracy: 0.7037
Epoch 10/10
1563/1563 [==============================] - 75s 48ms/step - loss: 0.6312 - accuracy: 0.7783 - val_loss: 0.8678 - val_accuracy: 0.7145
```

#Plot Training History:

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()
```



#Evaluate the Model:

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 4s - loss: 0.8678 - accuracy: 0.7145 - 4s/epoch - 13ms/step
```

#Data Augmentation: Apply techniques like rotation, shifting, and flipping to artificially increase the size and variability of your training dataset.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
)

datagen.fit(train_images)
```

```
history = model.fit(datagen.flow(train_images, train_labels, batch_size=32),
                    epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 106s 68ms/step - loss: 1.1932 - accuracy: 0.5791 - val_loss: 0.9156 - val_accuracy: 0.6895
Epoch 2/10
1563/1563 [==============================] - 108s 69ms/step - loss: 1.1251 - accuracy: 0.6029 - val_loss: 0.8705 - val_accuracy: 0.6991
Epoch 3/10
1563/1563 [==============================] - 103s 66ms/step - loss: 1.0981 - accuracy: 0.6125 - val_loss: 1.0285 - val_accuracy: 0.6477
Epoch 4/10
1563/1563 [==============================] - 106s 68ms/step - loss: 1.0640 - accuracy: 0.6267 - val_loss: 1.0016 - val_accuracy: 0.6535
Epoch 5/10
1563/1563 [==============================] - 103s 66ms/step - loss: 1.0415 - accuracy: 0.6322 - val_loss: 1.0259 - val_accuracy: 0.6570
Epoch 6/10
1563/1563 [==============================] - 105s 67ms/step - loss: 1.0202 - accuracy: 0.6421 - val_loss: 1.0373 - val_accuracy: 0.6513
Epoch 7/10
1563/1563 [==============================] - 107s 68ms/step - loss: 0.9983 - accuracy: 0.6476 - val_loss: 0.9541 - val_accuracy: 0.6777
Epoch 8/10
1563/1563 [==============================] - 106s 68ms/step - loss: 0.9818 - accuracy: 0.6523 - val_loss: 1.0051 - val_accuracy: 0.6716
Epoch 9/10
1563/1563 [==============================] - 106s 68ms/step - loss: 0.9800 - accuracy: 0.6555 - val_loss: 0.8848 - val_accuracy: 0.6959
Epoch 10/10
1563/1563 [==============================] - 106s 68ms/step - loss: 0.9674 - accuracy: 0.6607 - val_loss: 0.8827 - val_accuracy: 0.6968
```

#Fine-Tuning Pre-Trained Models: Use pre-trained models like VGG16, ResNet, or MobileNet for transfer learning to leverage the power of models trained on large datasets.

```
base_model = tf.keras.applications.VGG16(input_shape=(32, 32, 3),
                                         include_top=False,
                                         weights='imagenet')
base_model.trainable = False

model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

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