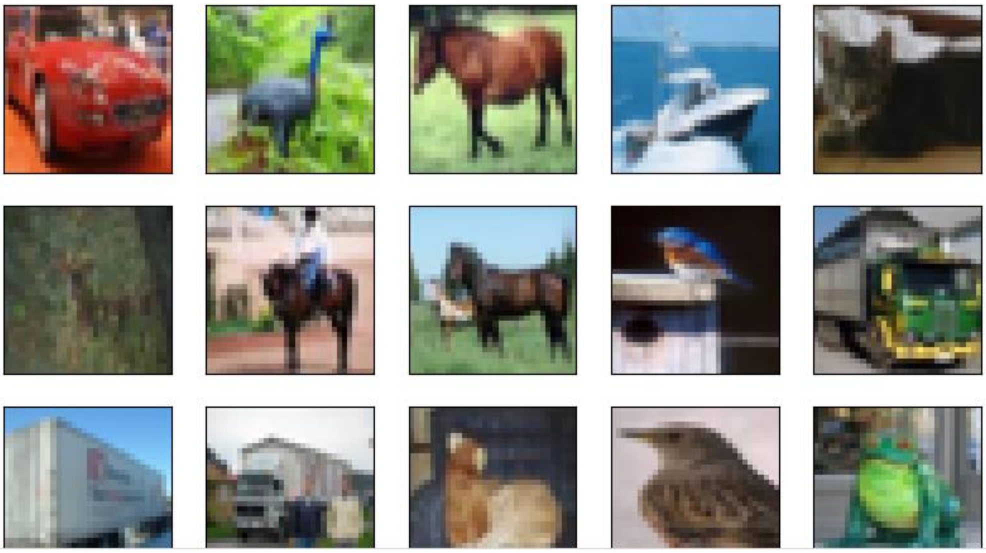
Program

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
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(train_images[i])

# The CIFAR labels happen to be arrays, which is why you need the extra index
plt.xlabel(class_names[train_labels[i][0]])
plt.show()
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.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
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))
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')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(test_acc)
```

Output

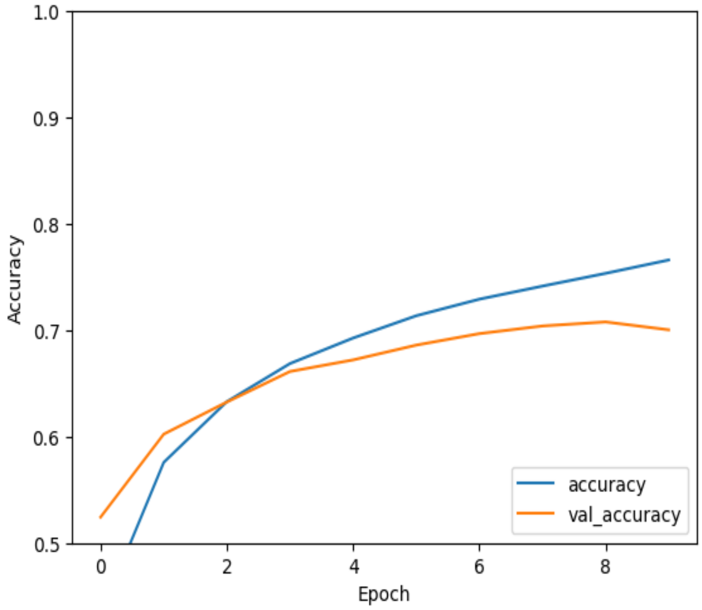


```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 64)                65600

 dense_3 (Dense)             (None, 10)                650

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

```
Epoch 1/10
1563/1563 [==============================] - 79s 49ms/step - loss: 1.5548 - accuracy: 0.4323 - val_loss: 1.3287 - val_accuracy: 0.5243
Epoch 2/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.1896 - accuracy: 0.5758 - val_loss: 1.1114 - val_accuracy: 0.6023
Epoch 3/10
1563/1563 [==============================] - 64s 41ms/step - loss: 1.0427 - accuracy: 0.6329 - val_loss: 1.0360 - val_accuracy: 0.6325
Epoch 4/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.9469 - accuracy: 0.6687 - val_loss: 0.9580 - val_accuracy: 0.6612
Epoch 5/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.8757 - accuracy: 0.6927 - val_loss: 0.9287 - val_accuracy: 0.6721
Epoch 6/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.8180 - accuracy: 0.7136 - val_loss: 0.9081 - val_accuracy: 0.6860
Epoch 7/10
1563/1563 [==============================] - 65s 41ms/step - loss: 0.7716 - accuracy: 0.7292 - val_loss: 0.8798 - val_accuracy: 0.6968
Epoch 8/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.7312 - accuracy: 0.7415 - val_loss: 0.8748 - val_accuracy: 0.7040
Epoch 9/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.6952 - accuracy: 0.7535 - val_loss: 0.8652 - val_accuracy: 0.7078
Epoch 10/10
1563/1563 [==============================] - 63s 41ms/step - loss: 0.6633 - accuracy: 0.7660 - val_loss: 0.8976 - val_accuracy: 0.7004
313/313 - 3s - loss: 0.8976 - accuracy: 0.7004 - 3s/epoch - 10ms/step
0.7003999948501587
```


```
313/313 - 3s - loss: 0.8976 - accuracy: 0.7004 - 3s/epoch - 10ms/step
0.7003999948501587
```

Advantages of Convolutional Neural Networks (CNNs)

- Good at detecting patterns and features in images, videos, and audio signals.
- Robust to translation, rotation, and scaling invariance.
- End-to-end training, no need for manual feature extraction.
- Can handle large amounts of data and achieve high accuracy.

Disadvantages of Convolutional Neural Networks (CNNs)

- Computationally expensive to train and require a lot of memory.
- Can be prone to overfitting if not enough data or proper regularization is used.
- Requires large amounts of labeled data.
- Interpretability is limited, it’s hard to understand what the network has learned.