INTERN:- Bhagyashri Sharad Pisal *Intern ID:- CT4MOTH

*Domain:- Machine Learning

*Duration:-January 25, 2025, to May 25, 2025

*Company:- CODETECH IT SOLUTIONS

*Mentor:- Neela Santhosh Kumar

TASK THREE: IMAGE CLASSIFICATION MODEL

BUILD A CONVOLUTIONAL NEURAL NETWORK (CNN) FOR IMAGE CLASSIFICATION USING TENSORFLOW OR PYTORCH.

DELIVERABLE: A FUNCTIONAL MODEL WITH PERFORMANCE EVALUATION ON A TEST DATASET.

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
```

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

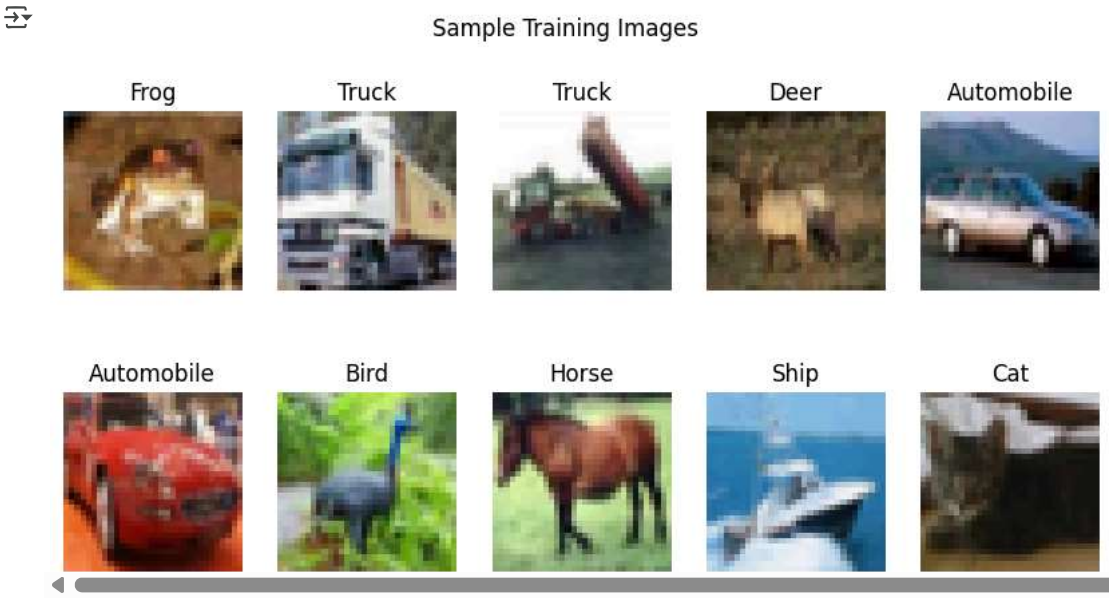
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
# Normalize the data to range [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

```
# Convert labels to one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
# Define class names
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']
```

```
# Visualize some training data
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_train[i])
    plt.title(class_names[int(tf.argmax(y_train[i]))])
    plt.axis('off')
plt.suptitle("Sample Training Images")
plt.show()
```




```
# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pr
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Generate randomly select 5 items from a list Close

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

```
# Display the model architecture
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65,600
dense_1 (Dense)	(None, 10)	650

Total params: 122,570 (478.79 KB)

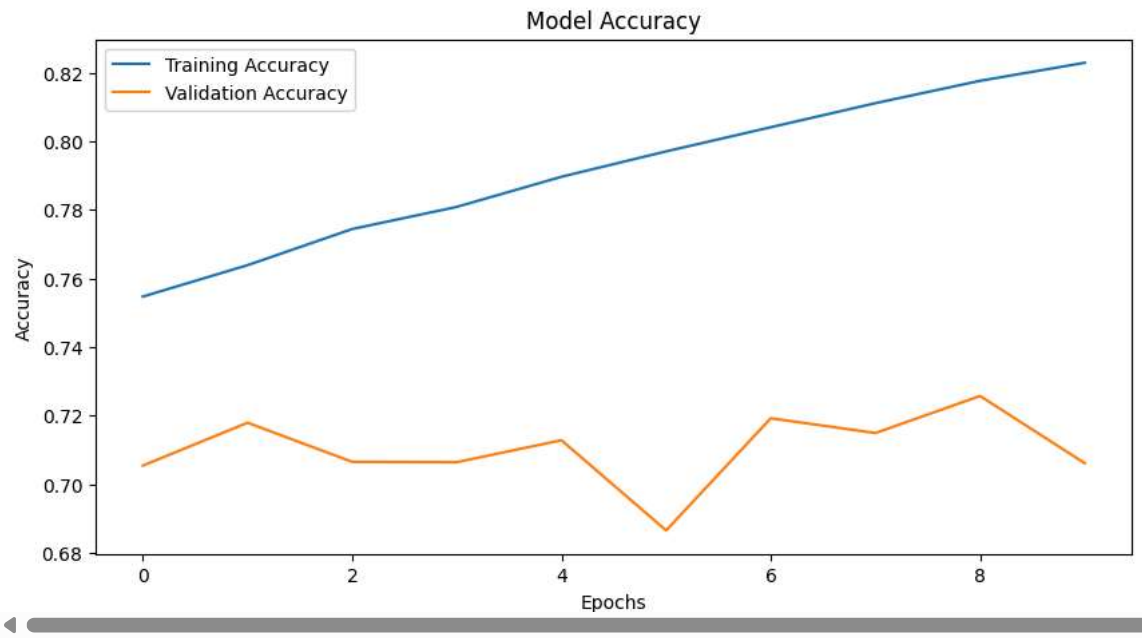
```
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.7553 - loss: 0.6971 - val_accuracy: 0.7054 - val_loss: 0.8698
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 81s 82ms/step - accuracy: 0.7658 - loss: 0.6666 - val_accuracy: 0.7179 - val_loss: 0.8358
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 63s 81ms/step - accuracy: 0.7785 - loss: 0.6328 - val_accuracy: 0.7065 - val_loss: 0.8437
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 83s 82ms/step - accuracy: 0.7841 - loss: 0.6175 - val_accuracy: 0.7064 - val_loss: 0.8744
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 67s 85ms/step - accuracy: 0.7945 - loss: 0.5915 - val_accuracy: 0.7128 - val_loss: 0.8554
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 65s 83ms/step - accuracy: 0.7993 - loss: 0.5724 - val_accuracy: 0.6865 - val_loss: 0.9622
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 83ms/step - accuracy: 0.8036 - loss: 0.5603 - val_accuracy: 0.7192 - val_loss: 0.8475
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 83ms/step - accuracy: 0.8169 - loss: 0.5235 - val_accuracy: 0.7149 - val_loss: 0.8662
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 83ms/step - accuracy: 0.8232 - loss: 0.5060 - val_accuracy: 0.7257 - val_loss: 0.8632
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 83ms/step - accuracy: 0.8276 - loss: 0.4908 - val_accuracy: 0.7061 - val_loss: 0.9219
```

```
# Evaluate the model on test data
loss, accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

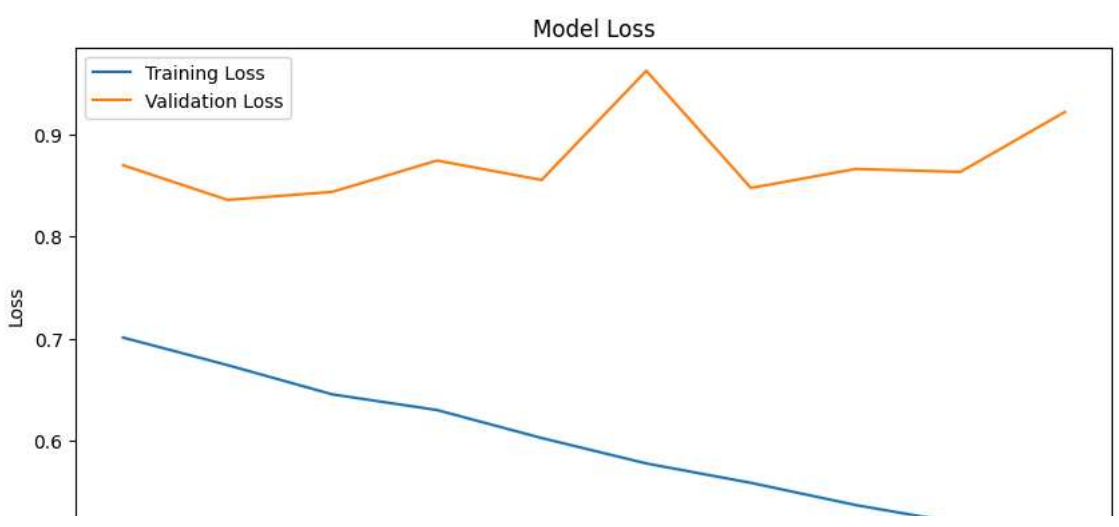
```
313/313 - 5s - 15ms/step - accuracy: 0.7061 - loss: 0.9219
Test Accuracy: 70.61%
```

```
# Plot training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Generate print hello world using rot13 Close

```
# Plot training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Model Loss
Training Loss
Validation Loss
Loss
0.9
0.8
0.7
0.6