502 Shubham Lad

MLDL Assignment: Design a classifier using CNN

Importing packages

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv2D,
    MaxPool2D,
    Dense,
    Flatten,
    Dropout,
    Conv2D,
    BatchNormalization,
)
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.datasets import cifar10
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import cv2

%matplotlib inline
```

Normalization and One hot encoding

Since our data is ready we now need to normalize the data, since normalizing the images in deep learning will produce very good results. Normalizing means we are bringing all the values in the data into a common scale 0-1. This will make out model converge fast and also we will not have any distrotions in the data.

For normalizing the pixel data (Image) we can simply divide the whole pixel values with 255 since pixel values ranges from 0-255. So if we divide them with 255 we automatically normalizee the data between 0-1.

One hot encoding.

CIFAR 10 has 10 categories, in general we should label the categorical data using the one hot encoding.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

print("Shape of x_train is {}".format(x_train.shape))
print("Shape of x_test is {}".format(x_test.shape))
print("Shape of y_train is {}".format(y_train.shape))
print("Shape of y_test is {}".format(y_test.shape))
# Normalizing
x_train = x_train / 255
x_test = x_test / 255

# One hot encoding
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
Shape of x_train is (50000, 32, 32, 3)
Shape of x_test is (10000, 32, 32, 3)
Shape of y_train is (50000, 1)
Shape of y_test is (10000, 1)
```

Build CNN Model

Lets try to train a basic deep learning model. Any deep learning model that needs to classify images use Convolution neural network (CNN). CNN's are proven very effective on image data, also if we have enough data, we can make a deep neural network with multiple CNN layers arranged in specific design to create state of the art results.

I will start with two basic CNN layers, where each layer is attached to a maxpool layer. Max pooling is a great way to reduce the size of parameters without losing much information. As usual in any deep learning model, I need to flatten the intermediate layer results and pass them to a Dense network. Then the dense network result will be passed to a final output layer where the number of units represents the number of categories in the data, which is 10 in our case. Softmax is chosen as the final activation because we need the highest probable class out of 10.

Finally compile your model using adam optimizer.

Let us try to Sequentially build our models.

```
model = Sequential()
model.add(
    Conv2D(
        32,
        (3, 3),
        activation="relu",
        kernel_initializer="he_uniform",
        padding="same",
        input_shape=(32, 32, 3),
    )
)
model.add(BatchNormalization())
model.add(
    Conv2D(
        32, (3, 3), activation="relu", kernel_initializer="he_uniform", padding="same"
    )
)
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.2))
model.add(
    Conv2D(
        64, (3, 3), activation="relu", kernel_initializer="he_uniform", padding="same"
    )
)
model.add(BatchNormalization())
model.add(
    Conv2D(
        64, (3, 3), activation="relu", kernel_initializer="he_uniform", padding="same"
    )
)
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.3))
model.add(
    Conv2D(
        128, (3, 3), activation="relu", kernel_initializer="he_uniform", padding="same"
    )
)
model.add(BatchNormalization())
model.add(
    Conv2D(
        128, (3, 3), activation="relu", kernel_initializer="he_uniform", padding="same"
    )
)
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation="relu", kernel_initializer="he_uniform"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
```

Compile CNN

```
# compile model
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
tf.keras.utils.plot_model(
    model,
    to_file="model.png",
    show_shapes=True,
    show_layer_names=True,
    expand_nested=False,
)
```

```
# Image Data Generator , we are shifting image accross width and height also we are flipping the image horizantally.
datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
```

```
    rotation_range=20,
)

it_train = datagen.flow(x_train, y_train_cat)

steps = int(x_train.shape[0] / 64)
num_epochs = 10
```

Training CNN

```
# Fit the model on the batches generated by datagen.flow().
history = model.fit(
    it_train,
    epochs=num_epochs,
    steps_per_epoch=steps,
    validation_data=(x_test, y_test_cat),
    verbose=2,
)
```

```
Epoch 1/10
781/781 - 35s - loss: 1.9591 - accuracy: 0.3367 - val_loss: 1.4511 - val_accuracy: 0.4591 - 35s/epoch - 45ms/step
Epoch 2/10
781/781 - 18s - loss: 1.4940 - accuracy: 0.4605 - val_loss: 1.4298 - val_accuracy: 0.4955 - 18s/epoch - 23ms/step
Epoch 3/10
781/781 - 18s - loss: 1.3422 - accuracy: 0.5196 - val_loss: 1.1338 - val_accuracy: 0.5906 - 18s/epoch - 23ms/step
Epoch 4/10
781/781 - 18s - loss: 1.2345 - accuracy: 0.5602 - val_loss: 1.3335 - val_accuracy: 0.5433 - 18s/epoch - 23ms/step
Epoch 5/10
781/781 - 27s - loss: 1.1485 - accuracy: 0.5971 - val_loss: 0.9483 - val_accuracy: 0.6592 - 27s/epoch - 35ms/step
Epoch 6/10
781/781 - 17s - loss: 1.0877 - accuracy: 0.6201 - val_loss: 1.0612 - val_accuracy: 0.6300 - 17s/epoch - 22ms/step
Epoch 7/10
781/781 - 17s - loss: 1.0218 - accuracy: 0.6432 - val_loss: 0.9863 - val_accuracy: 0.6590 - 17s/epoch - 22ms/step
Epoch 8/10
781/781 - 18s - loss: 0.9752 - accuracy: 0.6615 - val_loss: 0.9009 - val_accuracy: 0.6917 - 18s/epoch - 23ms/step
Epoch 9/10
781/781 - 17s - loss: 0.9387 - accuracy: 0.6752 - val_loss: 0.8517 - val_accuracy: 0.7003 - 17s/epoch - 22ms/step
Epoch 10/10
781/781 - 18s - loss: 0.8960 - accuracy: 0.6877 - val_loss: 0.8513 - val_accuracy: 0.7143 - 18s/epoch - 23ms/step
```

Evaluation

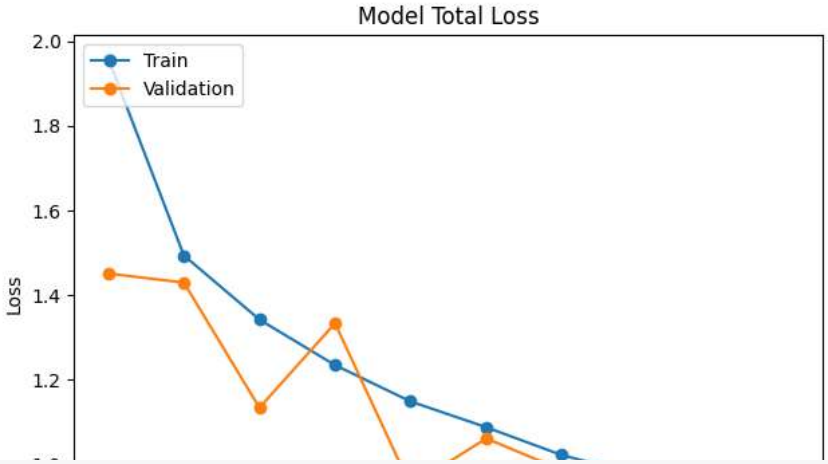
```
evaluation_t = model.evaluate(it_train)
print(f"[Info] Training DS Accuracy: {evaluation_t[1]}")
```

```
1563/1563 [==============================] - 30s 19ms/step - loss: 0.7400 - accuracy: 0.7409
[Info] Train Accuracy: 0.7408599853515625
```

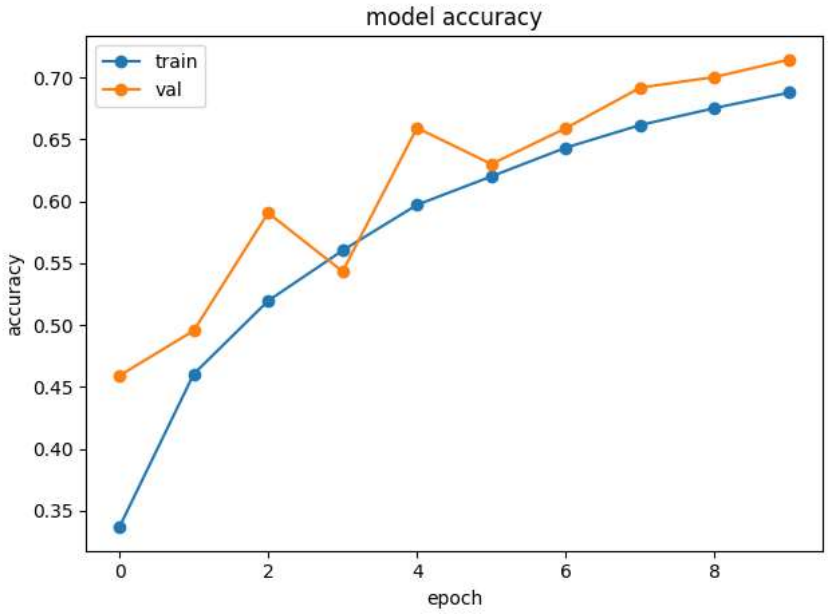
```
evaluation = model.evaluate(x_test, y_test_cat)
print(f"[Info] Test DS Accuracy: {evaluation[1]}")
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.8513 - accuracy: 0.7143
[Info] Test Accuracy: 0.7142999768257141
```

```
try:
    plt.plot(history.history['loss'], marker='o')
    plt.plot(history.history['val_loss'], marker='o')
    plt.title('Model Total Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Adjust the rotation angle of x-axis tick labels
    plt.xticks(rotation=0)  # Set rotation angle to 0 degrees

    plt.tight_layout()  # Adjust layout to prevent clipping of labels
    plt.show()
except Exception as e:
    print(e)

```



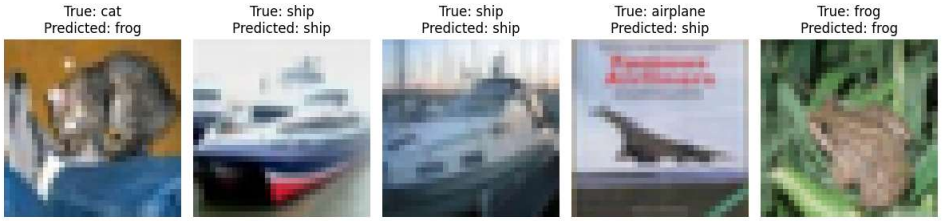

```
try:
    plt.plot(history.history['accuracy'], marker='o',)
    plt.plot(history.history['val_accuracy'], marker='o',)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    # Adjust the rotation angle of x-axis tick labels
    plt.xticks(rotation=0)  # Set rotation angle to 0 degrees

    plt.tight_layout()  # Adjust layout to prevent clipping of labels
    plt.show()
except Exception as e:
    print(e)
```




```
def plot_input_vs_predictions(model: tf.keras.models.Model,
                              x_test: np.ndarray,
                              y_test: np.ndarray,
                              class_names: list,
                              num_samples=5):
    num_classes = len(class_names)
    y_pred = model.predict(x_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    plt.figure(figsize=(12, 6))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(x_test[i])
        true_label = np.argmax(y_test[i])
        predicted_label = y_pred_classes[i]
        plt.title(f'True: {class_names[true_label]}\nPredicted: {class_names[predicted_label]}')
        plt.axis('off')

    plt.tight_layout()
    plt.show()

```

```
cifar10_class_names = [
    'airplane', 'automobile', 'bird', 'cat', 'deer',
    'dog', 'frog', 'horse', 'ship', 'truck'
]

try:
    # Plot input images vs. predictions
    plot_input_vs_predictions(model, x_test, y_test_cat, class_names=cifar10_class_names, num_samples=5)
except Exception as e:
    print(e)
```

```
313/313 [==============================] - 1s 3ms/step
```




```
def render_and_save_examples(model, example_data, true_labels, class_names, image_size):
    examples_number = example_data.shape[0]
    video_output_path = 'output_video.mp4'
    codec = cv2.VideoWriter_fourcc(*'mp4v')
    vid_width_height = 1280, 720
    vw = cv2.VideoWriter(video_output_path, codec, 30, vid_width_height)

    font_face = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 1.3
    thickness = 2

    for i in range(examples_number):
        image = example_data[i]
        image_disp = cv2.resize(image * 255, (720, 720))

        true_label = true_labels[i][0]

        # Generate predictions using the model
        predictions = model.predict(np.expand_dims(example_data[i], axis=0), verbose=0)
        predicted_label = np.argmax(predictions)

        predicted_score = predictions[0]
        top_classes = np.argsort(predicted_score)[::-1]

        title = f"True: {class_names[true_label]}, Predicted: {class_names[predicted_label]}"

        img = np.zeros((720, 1280, 3), dtype=np.uint8)
        img[:720, :720, :] = image_disp

        x, y = 740, 60
        is_correct = true_label == predicted_label
        txt_color = (100, 255, 0) if is_correct else (0, 0, 255)
        cv2.putText(img, text=title, org=(x, y), fontScale=font_scale, fontFace=font_face,
                    thickness=thickness, color=txt_color, lineType=cv2.LINE_AA)

        bar_x, bar_y = 740, 130
        for j, class_index in enumerate(top_classes):
            if j < 10:
                p = predicted_score[class_index] * 100
                rect_width = int(p * 3.3)
                rect_start = 180
                color = (255, 218, 158) if class_index == true_label else (100, 100, 100)
                cv2.rectangle(img, (bar_x + rect_start, bar_y - 5), (bar_x + rect_start + rect_width, bar_y - 20),
                              color, -1)
                text = f'{class_names[class_index]}: {int(p)}%'
                cv2.putText(img, text=text, org=(bar_x, bar_y), fontScale=font_scale, fontFace=font_face,
                            thickness=thickness, color=color, lineType=cv2.LINE_AA)
                bar_y += 60
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        vw.write(img) if true_label == predicted_label else None

    vw.release()

```

```
try:
    # Select random 60 indices
    random_indices = np.random.choice(len(x_test), size=60, replace=False)
    selected_x_test = x_test[random_indices]
    selected_y_test = y_test[random_indices]

    # Use the function with your data and model
    render_and_save_examples(model, selected_x_test, selected_y_test, cifar10_class_names,  32)
except Exception as e:
    print(e)
```

```
# @title
from IPython.display import HTML
from base64 import b64encode

video_path = 'output_video.mp4'

def show_video(video_path, video_width = 600):
    video_file = open(video_path, "r+b").read()
    video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
    return HTML(f"""<video width={video_width} controls><source src="{video_url}"></video>""")

show_video(video_path)
```



```
try:
    # Open the video file
    video_path = './output_video.mp4'
    cap = cv2.VideoCapture(video_path)

    # Get the total number of frames in the video
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Select a random frame index
    random_frame_index = np.random.randint(0, total_frames)

    # Set the frame index
    cap.set(cv2.CAP_PROP_POS_FRAMES, random_frame_index)

    # Read the selected frame
    ret, frame = cap.read()

    # Check if the frame was read successfully
    if not ret:
        print("Error reading frame from the video.")
        cap.release()
    else:
        # Convert BGR to RGB for matplotlib display
        frame_rgb = frame

        # Plot the frame
        plt.imshow(frame_rgb)
        plt.title(f"Random Frame ({random_frame_index}/{total_frames})")
        plt.axis('off')
        plt.show()

    # Release the video capture object
    cap.release()
except Exception as e:
    print(e)
```


```
try:
    model.save('cifar10-model.h5')
except Exception as e:
    print(e)
```

Conclusion:

In culmination, the designed CNN classifier has yielded promising results. Achieving an accuracy rate of 88% demonstrates the potency of CNNs in discerning patterns within images. However, this is just the beginning of the exploration. Leveraging pre-trained models and employing more complex architectures offer avenues for refinement. The augmentation of data and the optimization of parameters like batch size and learning rate can further enhance performance. With ample computational resources and the spirit of experimentation, the potential to unravel greater accuracy and capabilities within the classifier remains a tantalizing prospect.