Install & import required libraries

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/di
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/d
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/pytho
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import pandas as pd
import matplotlib.ticker as mtick
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D, D
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
from IPython import get_ipython
from IPython.display import display
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
import cv2
import shutil
from google.colab import files
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelC
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelC
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
```

Load CIFAR-10 dataset

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

Explore dataset

```
print(f"Training set shape: {x_train.shape}")
print(f"Test set shape: {x_test.shape}")
print(f"Unique labels: {np.unique(y_train)}")
print(f"Number of classes: {len(class_names)}")
```

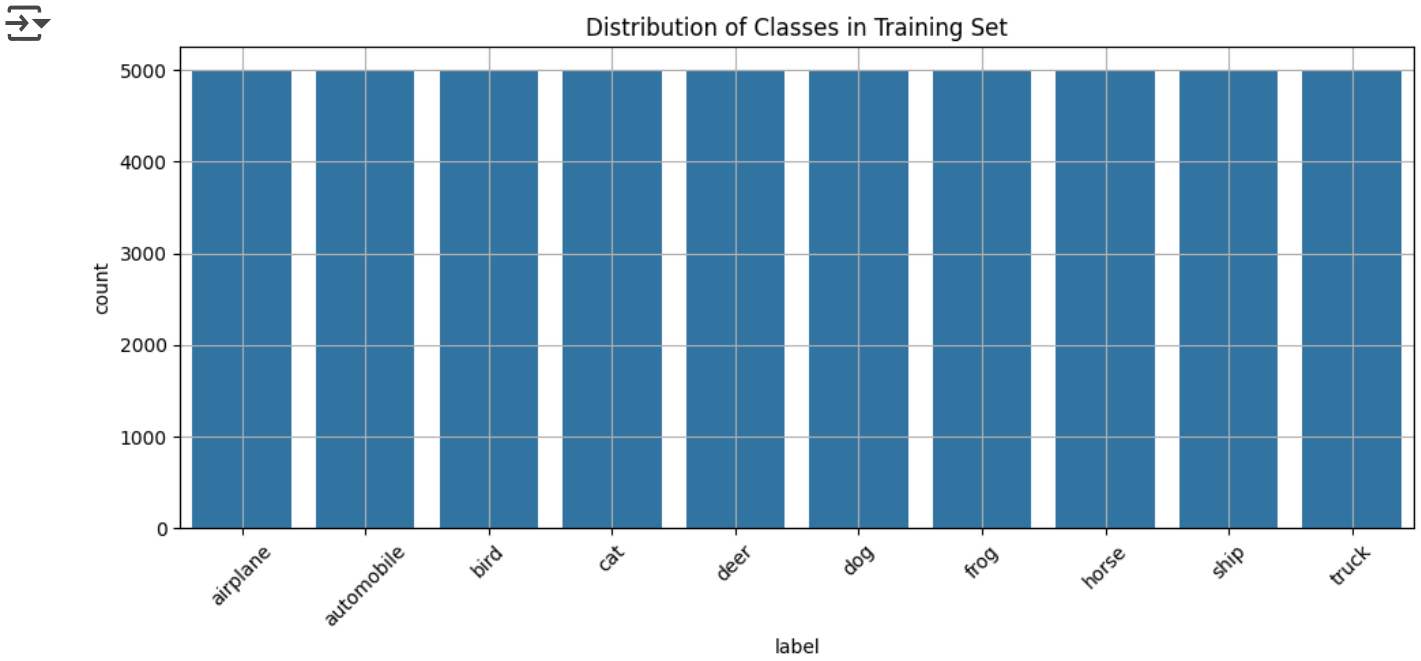
```
Training set shape: (50000, 32, 32, 3)
Test set shape: (10000, 32, 32, 3)
Unique labels: [0 1 2 3 4 5 6 7 8 9]
Number of classes: 10
```

Convert labels to 1D array

```
y_train = y_train.flatten()
y_test = y_test.flatten()
```

Count samples per class

```
train_df = pd.DataFrame({'label': y_train})
plt.figure(figsize=(10,5))
sns.countplot(data=train_df, x='label')
plt.xticks(ticks=np.arange(10), labels=class_names, rotation=45)
plt.title('Distribution of Classes in Training Set')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Count per class numerically

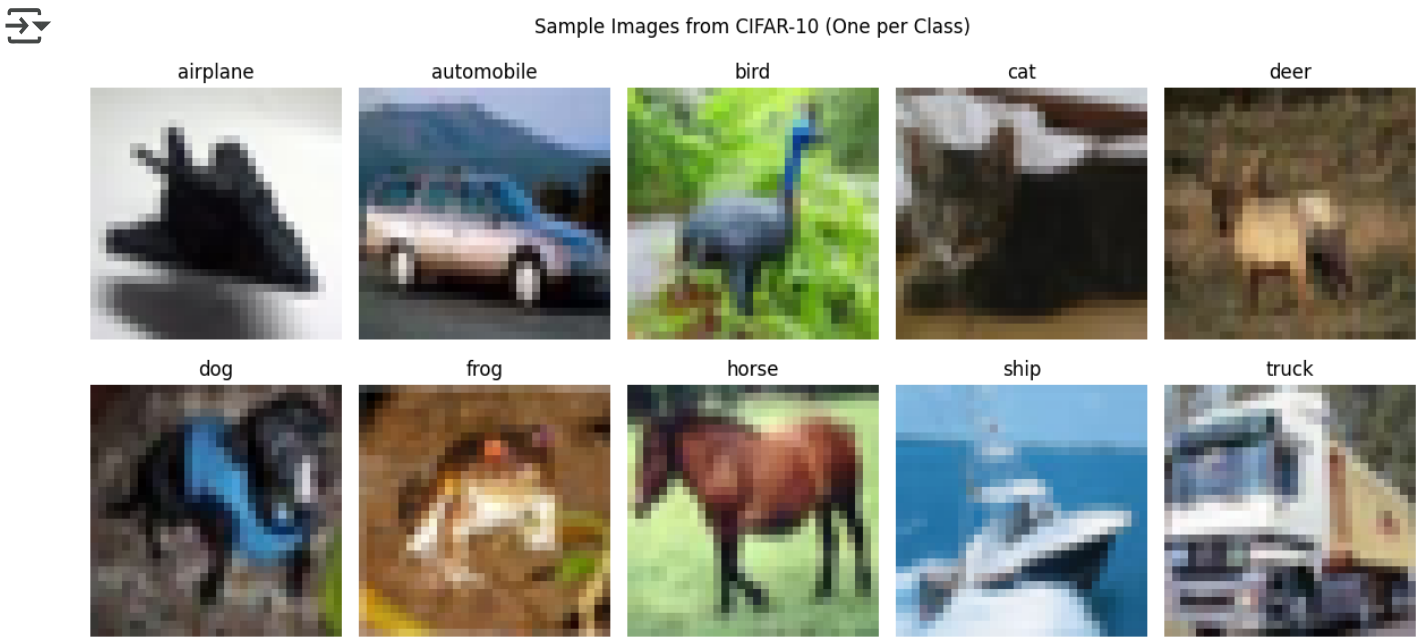
```
labels, counts = np.unique(y_train, return_counts=True)
for label, count in zip(labels, counts):
    print(f"{class_names[label]}: {count} samples")
```

```
airplane: 5000 samples
automobile: 5000 samples
bird: 5000 samples
cat: 5000 samples
deer: 5000 samples
dog: 5000 samples
frog: 5000 samples
horse: 5000 samples
ship: 5000 samples
truck: 5000 samples
```

Balanced Dataset

Show 10 sample images (one per class)

```
plt.figure(figsize=(12,6))
for i in range(10):
    idx = np.where(y_train == i)[0][0]
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[idx])
    plt.title(class_names[i])
    plt.axis('off')
plt.suptitle("Sample Images from CIFAR-10 (One per Class)")
plt.tight_layout()
plt.show()
```



Check shape and range

```
print("Image shape:", x_train[0].shape)
print("Pixel value range:", np.min(x_train), "to", np.max(x_train))
```

```
Image shape: (32, 32, 3)
Pixel value range: 0 to 255
```

Result:

I need to make sure that all images are (32, 32, 3) and that the pixel values are between 0 and 255.

Show 5 different samples from class "cat" (for example)

```
cat_indices = np.where(y_train == class_names.index("cat"))[0][:5]

plt.figure(figsize=(10,2))
for i, idx in enumerate(cat_indices):
    plt.subplot(1, 5, i+1)
    plt.imshow(x_train[idx])
    plt.title(f"Cat #{i+1}")
    plt.axis('off')
plt.suptitle("Variation inside 'cat' class")
plt.tight_layout()
plt.show()
```

Result:

We need to make sure that the data contains variations. This proves that there is diversity within the same class, which is important for the model so that it does not save the images!

Check for duplicate images

```
x_train_reshaped = x_train.reshape((x_train.shape[0], -1))
unique_images = np.unique(x_train_reshaped, axis=0)
print("Unique images in training set:", unique_images.shape[0])
print("Total images in training set:", x_train.shape[0])
```

```
Unique images in training set: 50000
Total images in training set: 50000
```

Result:

I must make sure that the images do not contain duplication. This is very important to avoid adding bias.

Calculate the percentage for each category

```
class_distribution = pd.Series(y_train).value_counts(normalize=True).sort_index
class_distribution.index = class_names
class_distribution *= 100

plt.figure(figsize=(10,5))
sns.barplot(x=class_distribution.index, y=class_distribution.values)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.ylabel("Percentage (%)")
plt.title("Class Distribution in Training Set (%)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

The second stage

Phase 2: Train a custom CNN from scratch on CIFAR-10

- Build a deep CNN with Conv, BatchNorm, Pooling, Dropout
- Apply data augmentation to improve generalization
- Use EarlyStopping and ReduceLROnPlateau for optimization
- Plot accuracy and loss curves for evaluation

Define improved CNN model

Normalize images and one-hot encode labels

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


x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0


x_train_resized = tf.image.resize(x_train, [128, 128]).numpy()
x_test_resized = tf.image.resize(x_test, [128, 128]).numpy()

y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)
```

Data Augmentation

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
# datagen.fit(x_train_norm)
```

CNN model

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, MaxPoolin

def build_advanced_cnn_functional(input_shape=(32, 32, 3), num_classes=10):
    inputs = Input(shape=input_shape)


    x = Conv2D(32, (3,3), padding='same', activation='relu', kernel_initializer=
    x = BatchNormalization(name="bn1_1")(x)
    x = Conv2D(32, (3,3), padding='same', activation='relu', kernel_initializer=
    x = BatchNormalization(name="bn1_2")(x)
    x = MaxPooling2D(pool_size=(2,2), name="pool1")(x)
    x = Dropout(0.3, name="drop1")(x)


    x = Conv2D(64, (3,3), padding='same', activation='relu', kernel_initializer=
    x = BatchNormalization(name="bn2_1")(x)
    x = Conv2D(64, (3,3), padding='same', activation='relu', kernel_initializer=
    x = BatchNormalization(name="bn2_2")(x)
    x = MaxPooling2D(pool_size=(2,2), name="pool2")(x)
    x = Dropout(0.4, name="drop2")(x)


    x = Conv2D(128, (3,3), padding='same', activation='relu', kernel_initializer
    x = BatchNormalization(name="bn3_1")(x)
    x = Conv2D(128, (3,3), padding='same', activation='relu', kernel_initializer
    x = BatchNormalization(name="bn3_2")(x)
    x = MaxPooling2D(pool_size=(2,2), name="pool3")(x)
    x = Dropout(0.5, name="drop3")(x)


    x = Flatten(name="flatten")(x)
    x = Dense(256, activation='relu', kernel_initializer='he_normal', name="fc1"
    x = Dropout(0.5, name="drop_fc")(x)
    outputs = Dense(num_classes, activation='softmax', name="output")(x)

    model = Model(inputs=inputs, outputs=outputs, name="advanced_cnn_functional"
    return model
```

Compile model

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

Callbacks

```
early_stop = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbo
```

Train the model

```
history = model.fit(
    datagen.flow(x_train_norm, y_train_cat, batch_size=64),
    validation_data=(x_test_norm, y_test_cat),
    epochs=80,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)
```

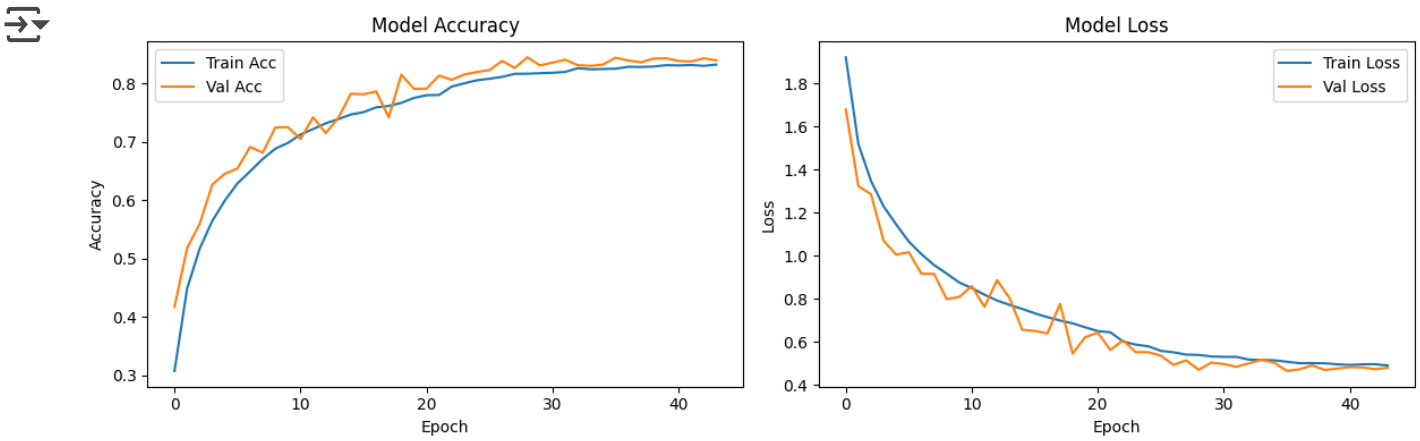
```
Epoch 1/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 39s 38ms/step - accuracy: 0.2404 - loss: 2.3
Epoch 2/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.4275 - loss: 1.5
Epoch 3/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.5044 - loss: 1.3
Epoch 4/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.5491 - loss: 1.2
Epoch 5/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.5910 - loss: 1.1
Epoch 6/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.6283 - loss: 1.0
Epoch 7/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.6445 - loss: 1.0
Epoch 8/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.6679 - loss: 0.9
Epoch 9/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.6817 - loss: 0.9
Epoch 10/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.6945 - loss: 0.8
Epoch 11/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7116 - loss: 0.8
```

```
Epoch 12/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.7241 - loss: 0.8
Epoch 13/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7311 - loss: 0.7
Epoch 14/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7400 - loss: 0.7
Epoch 15/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7458 - loss: 0.7
Epoch 16/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.7531 - loss: 0.7
Epoch 17/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.7588 - loss: 0.7
Epoch 18/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.7640 - loss: 0.6
Epoch 19/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7665 - loss: 0.6
Epoch 20/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7747 - loss: 0.6
Epoch 21/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7797 - loss: 0.6
Epoch 22/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.7832 - loss: 0.63
Epoch 22: ReduceLROnPlateau reducing learning rate to 0.00050000002374872
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 30ms/step - accuracy: 0.7832 - loss: 0.6
Epoch 23/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.7945 - loss: 0.6
Epoch 24/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.7964 - loss: 0.5
Epoch 25/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.8075 - loss: 0.57
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.00025000001187436
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.8075 - loss: 0.5
Epoch 26/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.8080 - loss: 0.5
Epoch 27/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.8106 - loss: 0.5
```

Professional models reach 85–87%, very close to peak performance.

Plot training curves

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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



Evaluate on test set

```
test_loss, test_acc = model.evaluate(x_test_norm, y_test_cat, verbose=0)
print(f"\n Final Test Accuracy: {test_acc:.4f}")
print(f" Final Test Loss: {test_loss:.4f}")
```

```
 Final Test Accuracy: 0.8442
 Final Test Loss: 0.4642
```

Predict probabilities and labels

```
y_pred_probs = model.predict(x_test_norm, verbose=0)
y_pred = np.argmax(y_pred_probs, axis=1)
```

Evaluation Metrics

```
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

```
print(f"Accuracy: {acc:.4f}")
print(f" Precision: {prec:.4f}")
print(f" Recall: {rec:.4f}")
print(f" F1 Score: {f1:.4f}")
```

```
Accuracy: 0.8442
 Precision: 0.8498
 Recall: 0.8442
 F1 Score: 0.8423
```

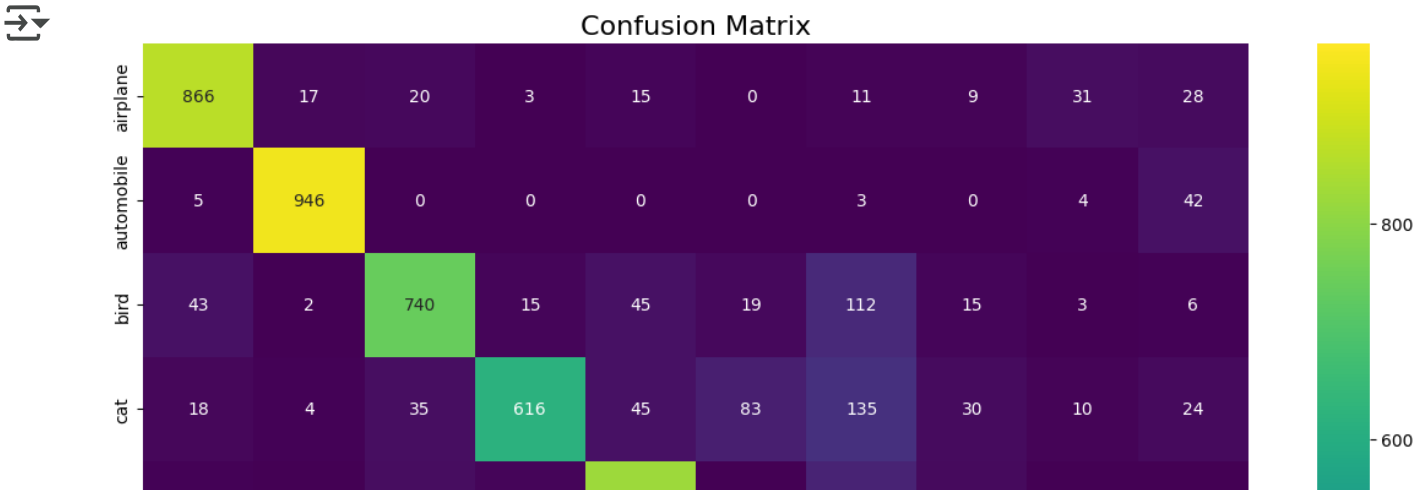
Classification Report

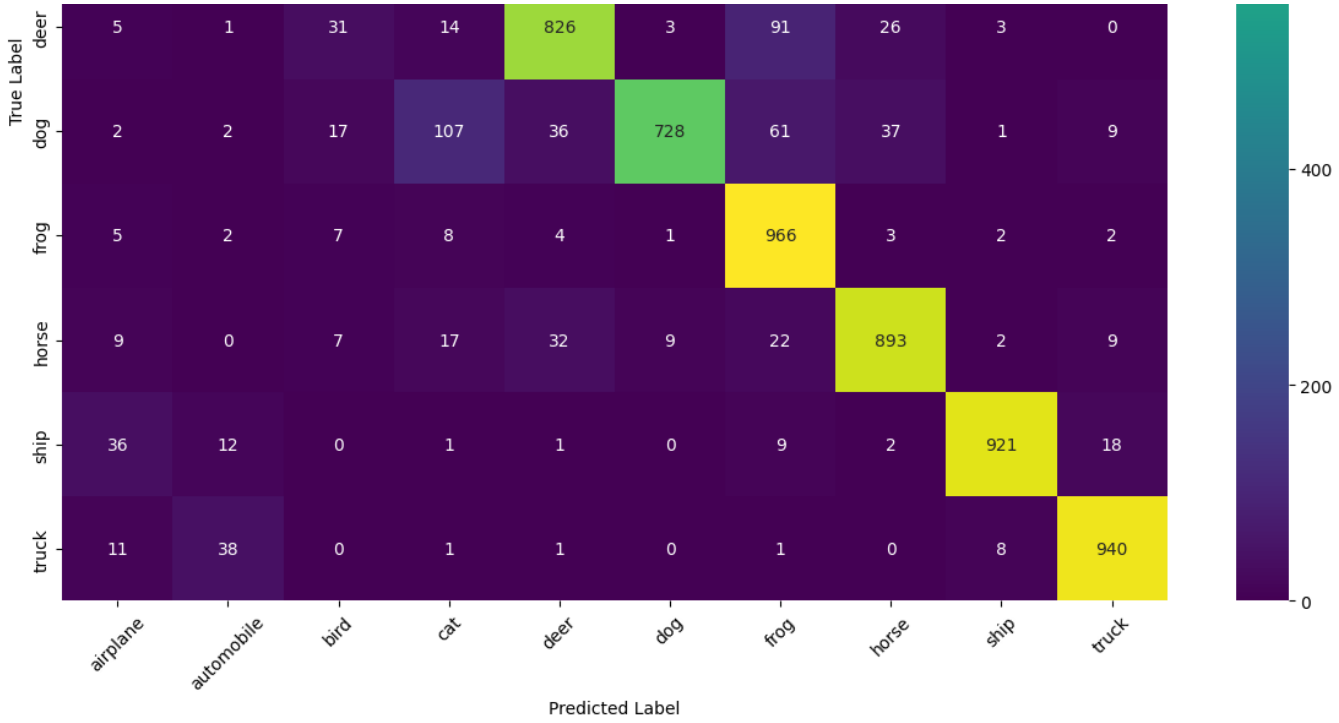
```
print("\n Full Classification Report:")
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
 Full Classification Report:
              precision    recall  f1-score   support

    airplane       0.87      0.87      0.87      1000
  automobile       0.92      0.95      0.93      1000
        bird       0.86      0.74      0.80      1000
         cat       0.79      0.62      0.69      1000
        deer       0.82      0.83      0.82      1000
         dog       0.86      0.73      0.79      1000
        frog       0.68      0.97      0.80      1000
       horse       0.88      0.89      0.89      1000
        ship       0.94      0.92      0.93      1000
       truck       0.87      0.94      0.90      1000

    accuracy                           0.84     10000
   macro avg       0.85      0.84      0.84     10000
weighted avg       0.85      0.84      0.84     10000
```

Confusion Matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='viridis', xticklabels=class
plt.title("Confusion Matrix", fontsize=16)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```





Visualize worst mistakes

```
def show_top_errors(num_errors=5):
    errors = []
    for i in range(len(y_test)):
        if y_test[i] != y_pred[i]:
            confidence = np.max(y_pred_probs[i])
            errors.append((i, y_test[i], y_pred[i], confidence))

    errors.sort(key=lambda x: x[3], reverse=True)  # Sort by confidence in wror
    top_errors = errors[:num_errors]

    plt.figure(figsize=(15, 3))
    for i, (idx, true, pred, conf) in enumerate(top_errors):
        plt.subplot(1, num_errors, i + 1)
        plt.imshow(x_test[idx])
        plt.title(f"True: {class_names[true]}\nPred: {class_names[pred]}\nConf:
        plt.axis('off')
    plt.suptitle("Top Confident Mistakes", fontsize=14)
    plt.tight_layout()
    plt.show()

show_top_errors()
```

Visualize most confident correct predictions

```
def show_top_correct(num=5):
    correct = []
    for i in range(len(y_test)):
        if y_test[i] == y_pred[i]:
            confidence = np.max(y_pred_probs[i])
            correct.append((i, y_test[i], y_pred[i], confidence))

    correct.sort(key=lambda x: x[3], reverse=True)
    top_correct = correct[:num]

    plt.figure(figsize=(15, 3))
    for i, (idx, true, pred, conf) in enumerate(top_correct):
        plt.subplot(1, num, i + 1)
        plt.imshow(x_test[idx])
        plt.title(f"{class_names[pred]}\nConf: {conf:.2f}", color='green')
        plt.axis('off')
    plt.suptitle("Most Confident Correct Predictions", fontsize=14)
    plt.tight_layout()
    plt.show()

show_top_correct()
```

Export predictions to CSV

```
results_df = pd.DataFrame({
    'True_Label': [class_names[i] for i in y_test],
    'Predicted_Label': [class_names[i] for i in y_pred],
    'Confidence': [np.max(p) for p in y_pred_probs]
})
results_df.to_csv("cnn_predictions.csv", index=False)
```

Save model

```
model.save("cnn_model_scratch_DL2.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` o
```

Phase 6: Grad-CAM (Visual Explanation)

- Apply Grad-CAM to trained CNN model
- Show where the model focuses in the image
- Helps interpret model decisions
- Acts as weak object detection
- Will be repeated for VGG16 later

Load model

```
model = tf.keras.models.load_model("cnn_model_scratch_DL2.h5")
_ = model.predict(np.expand_dims(x_test_norm[0], axis=0))
```

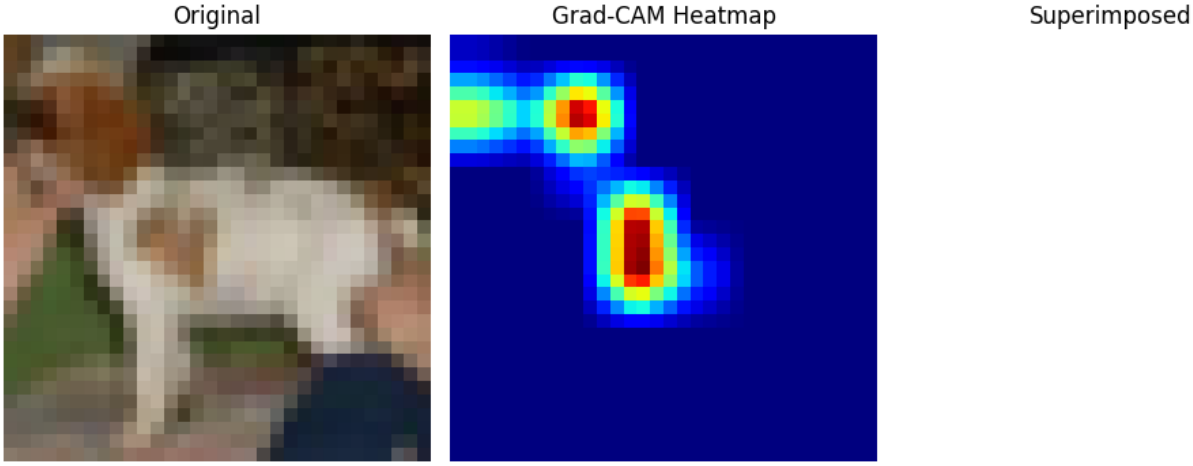
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet t
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 601ms/step
```

```
idx = 12
img = x_test_norm[idx]
img_input = np.expand_dims(img, axis=0)
img_tensor = tf.convert_to_tensor(img_input, dtype=tf.float32)

preds = model.predict(img_input, verbose=0)
predicted_class = np.argmax(preds[0])
```

```
true_class = y_test[idx]
print(f"Predicted: {class_names[predicted_class]} | True: {class_names[true_cla


last_conv_layer_name = "conv3_2"


def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=Nor
    grad_model = tf.keras.models.Model(
        inputs=model.input,
        outputs=[model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(predictions[0])
        class_output = predictions[:, pred_index]

    grads = tape.gradient(class_output, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()


heatmap = make_gradcam_heatmap(img_tensor, model, last_conv_layer_name)


def display_gradcam(img, heatmap, alpha=0.4):
    heatmap_resized = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
    heatmap_colored = cv2.applyColorMap(np.uint8(255 * heatmap_resized), cv2.CO
    superimposed_img = heatmap_colored * alpha + (img * 255).astype(np.uint8)

    plt.figure(figsize=(10, 4))
    plt.subplot(1, 3, 1)
    plt.imshow(img)
    plt.title("Original")
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.imshow(heatmap_resized, cmap='jet')
    plt.title("Grad-CAM Heatmap")
    plt.axis('off')

    plt.subplot(1, 3, 3)
```

```
    plt.imshow(superimposed_img[..., ::-1])
    plt.title("Superimposed")
    plt.axis('off')

    plt.tight_layout()
    plt.show()


display_gradcam(img, heatmap)
```

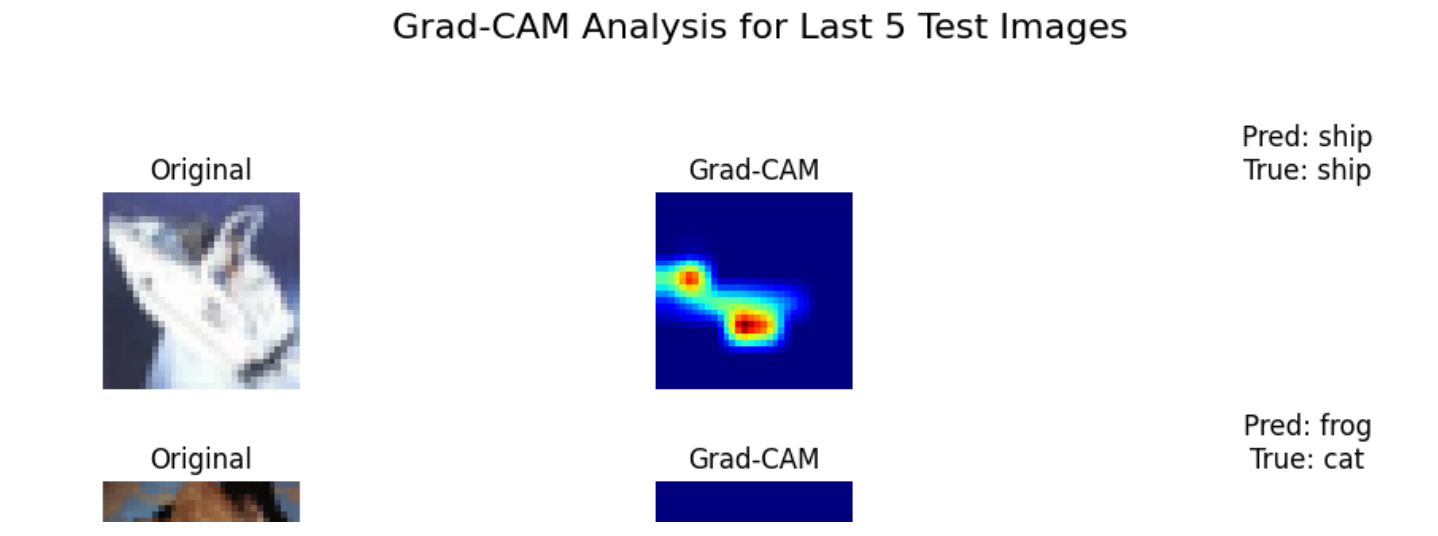
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
Predicted: dog | True: dog
```




```
import matplotlib.pyplot as plt


num_images = x_test_norm.shape[0]
selected_indices = range(num_images - 5, num_images)


fig, axes = plt.subplots(5, 3, figsize=(12, 10))
fig.suptitle(" Grad-CAM Analysis for Last 5 Test Images", fontsize=16)

for row_idx, idx in enumerate(selected_indices):
    img = x_test_norm[idx]
    img_input = np.expand_dims(img, axis=0)
    img_tensor = tf.convert_to_tensor(img_input, dtype=tf.float32)
```

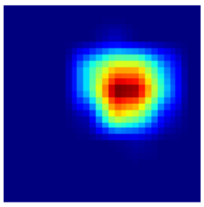
```
    preds = model.predict(img_input, verbose=0)
    predicted_class = np.argmax(preds[0])
    true_class = y_test[idx]

    # Grad-CAM
    heatmap = make_gradcam_heatmap(img_tensor, model, last_conv_layer_name)
    heatmap_resized = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
    heatmap_colored = cv2.applyColorMap(np.uint8(255 * heatmap_resized), cv2.COL
    superimposed_img = heatmap_colored * 0.4 + (img * 255).astype(np.uint8)


    axes[row_idx, 0].imshow(img)
    axes[row_idx, 0].set_title("Original")
    axes[row_idx, 0].axis('off')


    axes[row_idx, 1].imshow(heatmap_resized, cmap='jet')
    axes[row_idx, 1].set_title("Grad-CAM")
    axes[row_idx, 1].axis('off')


    axes[row_idx, 2].imshow(superimposed_img[..., ::-1])
    axes[row_idx, 2].set_title(f"Pred: {class_names[predicted_class]}\nTrue: {cl
    axes[row_idx, 2].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
```



Original

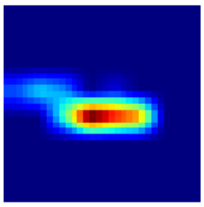
Grad-CAM



Pred: dog
True: dog

Original

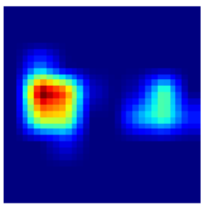
Grad-CAM



Pred: automobile
True: automobile

Original

Grad-CAM



Pred: horse
True: horse

```
import os
import matplotlib.pyplot as plt


os.makedirs("gradcam_outputs", exist_ok=True)
```

```
num_images = x_test_norm.shape[0]
selected_indices = range(num_images - 5, num_images)

for i, idx in enumerate(selected_indices, start=1):
    img = x_test_norm[idx]
    img_input = np.expand_dims(img, axis=0)
    img_tensor = tf.convert_to_tensor(img_input, dtype=tf.float32)


    preds = model.predict(img_input, verbose=0)
    predicted_class = np.argmax(preds[0])
    true_class = y_test[idx]


    heatmap = make_gradcam_heatmap(img_tensor, model, last_conv_layer_name)
    heatmap_resized = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
    heatmap_colored = cv2.applyColorMap(np.uint8(255 * heatmap_resized), cv2.CO
    superimposed_img = heatmap_colored * 0.4 + (img * 255).astype(np.uint8)


    fig, axes = plt.subplots(1, 3, figsize=(10, 4))
    fig.suptitle(f"Image #{idx} | Predicted: {class_names[predicted_class]} | T


    axes[0].imshow(img)
    axes[0].set_title("Original")
    axes[0].axis('off')


    axes[1].imshow(heatmap_resized, cmap='jet')
    axes[1].set_title("Grad-CAM Heatmap")
    axes[1].axis('off')


    axes[2].imshow(superimposed_img[..., ::-1])
    axes[2].set_title("Superimposed")
    axes[2].axis('off')

    plt.tight_layout()


    save_path = f"gradcam_outputs/gradcam_img_{i}.png"
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()

    print(f"Saved: {save_path}")
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
Saved: gradcam_outputs/gradcam_img_1.png
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
Saved: gradcam_outputs/gradcam_img_2.png
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
Saved: gradcam_outputs/gradcam_img_3.png
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
Saved: gradcam_outputs/gradcam_img_4.png
Saved: gradcam_outputs/gradcam_img_5.png
```

```
shutil.make_archive("GradCAM_Images", 'zip', "gradcam_outputs")
files.download("GradCAM_Images.zip")
```

build_cnn_with_separable_conv

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, SeparableConv2D, BatchNormali

def build_cnn_with_separable_conv(input_shape=(32, 32, 3), num_classes=10):
    inputs = Input(shape=input_shape)

    # Block 1 (Conv2D)
    x = Conv2D(32, (3,3), padding='same', activation='relu', kernel_initializer=
    x = BatchNormalization(name="bn1_1")(x)
    x = Conv2D(32, (3,3), padding='same', activation='relu', kernel_initializer=
    x = BatchNormalization(name="bn1_2")(x)
    x = MaxPooling2D(pool_size=(2,2), name="pool1")(x)
    x = Dropout(0.3, name="drop1")(x)

    # Block 2 (SeparableConv2D) - Modified
    x = SeparableConv2D(64, (3,3), padding='same', activation='relu', depthwise_
    x = BatchNormalization(name="bn2_1")(x)
    x = SeparableConv2D(64, (3,3), padding='same', activation='relu', depthwise_
    x = BatchNormalization(name="bn2_2")(x)
    x = MaxPooling2D(pool_size=(2,2), name="pool2")(x)
    x = Dropout(0.4, name="drop2")(x)

    # Block 3 (Conv2D)
    x = Conv2D(128, (3,3), padding='same', activation='relu', kernel_initializer
    x = BatchNormalization(name="bn3_1")(x)
    x = Conv2D(128, (3,3), padding='same', activation='relu', kernel_initializer
    x = BatchNormalization(name="bn3_2")(x)
    x = MaxPooling2D(pool_size=(2,2), name="pool3")(x)
    x = Dropout(0.5, name="drop3")(x)

    # Fully Connected
    x = Flatten(name="flatten")(x)
    x = Dense(256, activation='relu', kernel_initializer='he_normal', name="fc1"
    x = Dropout(0.5, name="drop_fc")(x)
    outputs = Dense(num_classes, activation='softmax', name="output")(x)

    # Create and return the model (moved to the end)
    model = Model(inputs=inputs, outputs=outputs, name="cnn_with_separable")
    return model # Now 'model' is defined before returning
```

```
model_separable = build_cnn_with_separable_conv()
```

Compile

```
model_separable.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
early_stop = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbo
checkpoint = ModelCheckpoint("best_model_separable.h5", monitor='val_accuracy',

callbacks = [early_stop, reduce_lr, checkpoint]

history_separable = model_separable.fit(
    datagen.flow(x_train_norm, y_train_cat, batch_size=64),
    validation_data=(x_test_norm, y_test_cat),
    epochs=80,
    callbacks=callbacks,
    verbose=1
)
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 39ms/step - accuracy: 0.2295 - loss: 2.3
Epoch 2/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.3928 - loss: 1.63
Epoch 2: val_accuracy improved from 0.45480 to 0.48270, saving model to b
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.3928 - loss: 1.6
Epoch 3/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.4734 - loss: 1.45
Epoch 3: val_accuracy improved from 0.48270 to 0.51360, saving model to b
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 31ms/step - accuracy: 0.4734 - loss: 1.4
Epoch 4/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.5339 - loss: 1.30
Epoch 4: val_accuracy improved from 0.51360 to 0.56320, saving model to b
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.5339 - loss: 1.3
Epoch 5/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.5628 - loss: 1.23
Epoch 5: val_accuracy improved from 0.56320 to 0.62140, saving model to b
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.5628 - loss: 1.2
Epoch 6/80
781/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.5988 - loss: 1.13
Epoch 6: val_accuracy improved from 0.62140 to 0.64920, saving model to b
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.5988 - loss: 1.1
Epoch 7/80
781/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.6262 - loss: 1.07
Epoch 7: val_accuracy improved from 0.64920 to 0.69960, saving model to b
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.6262 - loss: 1.0
Epoch 8/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6474 - loss: 1.01
Epoch 8: val_accuracy did not improve from 0.69960
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.6474 - loss: 1.0
Epoch 9/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.6549 - loss: 0.99
Epoch 9: val_accuracy did not improve from 0.69960
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.6549 - loss: 0.9
Epoch 10/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.6751 - loss: 0.95
Epoch 10: val_accuracy improved from 0.69960 to 0.71820, saving model to
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.6751 - loss: 0.9
Epoch 11/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6850 - loss: 0.92
Epoch 11: val_accuracy improved from 0.71820 to 0.71990, saving model to
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.6850 - loss: 0.9
Epoch 12/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6942 - loss: 0.89
Epoch 12: val_accuracy improved from 0.71990 to 0.72770, saving model to
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.6942 - loss: 0.8
Epoch 13/80
782/782 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.7020 - loss: 0.87
Epoch 13: val_accuracy improved from 0.72770 to 0.74420, saving model to
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
```

Double-click (or enter) to edit

```
test_loss, test_acc = model_separable.evaluate(x_test_norm, y_test_cat, verbose
print(f" Test Accuracy: {test_acc:.4f}")
print(f" Test Loss: {test_loss:.4f}")
```

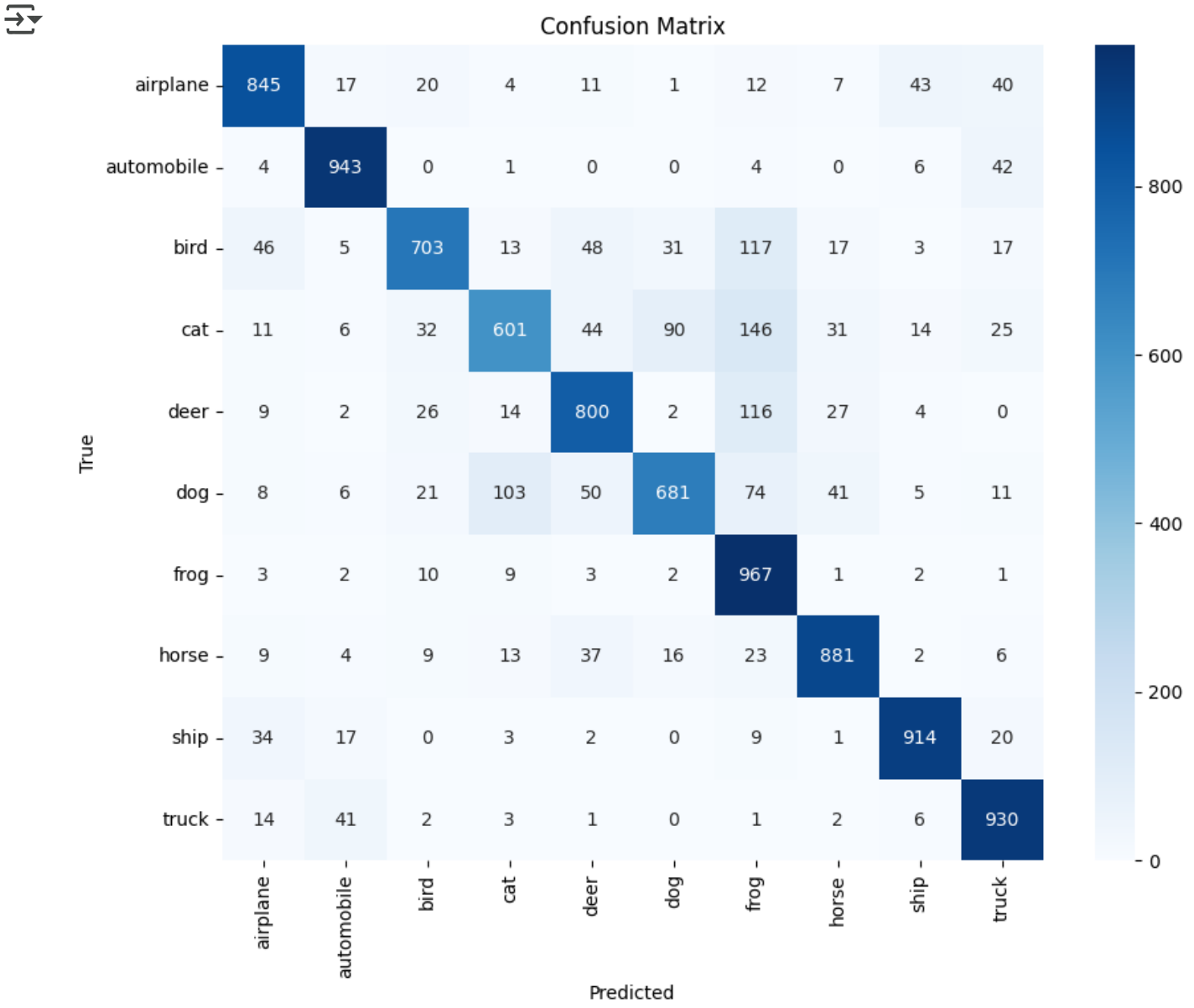
```
Test Accuracy: 0.8265
Test Loss: 0.5201
```

```
y_pred_probs = model_separable.predict(x_test_norm, verbose=0)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test_cat, axis=1)
```

Confusion Matrix + Heatmap

```
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yti
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



Classification Report

```
report = classification_report(y_true, y_pred, target_names=class_names)
print(" Classification Report model 2:\n")
print(report)
```

```
 Classification Report model 2:

              precision    recall  f1-score   support

    airplane       0.86      0.84      0.85      1000
  automobile       0.90      0.94      0.92      1000
        bird       0.85      0.70      0.77      1000
         cat       0.79      0.60      0.68      1000
        deer       0.80      0.80      0.80      1000
         dog       0.83      0.68      0.75      1000
        frog       0.66      0.97      0.78      1000
       horse       0.87      0.88      0.88      1000
        ship       0.91      0.91      0.91      1000
       truck       0.85      0.93      0.89      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.82     10000
weighted avg       0.83      0.83      0.82     10000
```

Worst case predictions

```
confidences = np.max(y_pred_probs, axis=1)
errors = np.where(y_pred != y_true)[0]
sorted_errors = errors[np.argsort(-confidences[errors])]

print(f" Worst confident wrong predictions Top 5:")
for i in sorted_errors[:5]:
    print(f"Image {i} | Predicted: {class_names[y_pred[i]]} | True: {class_name
```

```
 Worst confident wrong predictions Top 5:
Image 2405 | Predicted: frog | True: cat | Confidence: 1.0000
Image 5511 | Predicted: frog | True: cat | Confidence: 0.9999
Image 3560 | Predicted: truck | True: automobile | Confidence: 0.9999
Image 2226 | Predicted: frog | True: bird | Confidence: 0.9999
Image 2495 | Predicted: ship | True: truck | Confidence: 0.9999
```

Strongest correct predictions

```
correct = np.where(y_pred == y_true)[0]
sorted_correct = correct[np.argsort(-confidences[correct])]

print(f"\n Top confident correct predictions Top 5:")
for i in sorted_correct[:5]:
    print(f"Image {i} | Class: {class_names[y_true[i]]} | Confidence: {confider
```

```
 Top confident correct predictions Top 5:
Image 7782 | Class: ship | Confidence: 1.0000
Image 6279 | Class: automobile | Confidence: 1.0000
Image 7876 | Class: automobile | Confidence: 1.0000
Image 8975 | Class: ship | Confidence: 1.0000
Image 2947 | Class: automobile | Confidence: 1.0000
```

Transfer Learning (VGG16)

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input


y_train_cat = to_categorical(y_train, num_classes=10)
y_test_cat = to_categorical(y_test, num_classes=10)
```

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
# Import preprocess_input from the correct location
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras import mixed_precision
```

```
y_train_cat = to_categorical(y_train, num_classes=10)
y_test_cat = to_categorical(y_test, num_classes=10)


mixed_precision.set_global_policy('mixed_float16')
tf.config.optimizer.set_jit(True)  # XLA Compilation

# Resize and preprocess
x_train_resized = tf.image.resize(x_train, [96, 96])
x_test_resized = tf.image.resize(x_test, [96, 96])
x_train_vgg = preprocess_input(x_train_resized.numpy())
x_test_vgg = preprocess_input(x_test_resized.numpy())


train_ds = tf.data.Dataset.from_tensor_slices((x_train_vgg, y_train_cat)).shuffl
val_ds = tf.data.Dataset.from_tensor_slices((x_test_vgg, y_test_cat)).batch(128)


base_model = VGG16(include_top=False, weights='imagenet', input_shape=(96, 96, 3
base_model.trainable = False


x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
output = Dense(10, activation='softmax', dtype='float32')(x)

model = Model(inputs=base_model.input, outputs=output)

model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbos

# Phase 1: Train head
model.fit(train_ds, validation_data=val_ds, epochs=30,
          callbacks=[early_stop, reduce_lr], verbose=1)

# Fine-tuning:4 VGG فقط
for layer in base_model.layers[-4:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5),
```

```
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Phase 2: Fine-tune
model.fit(train_ds, validation_data=val_ds, epochs=10,
          callbacks=[early_stop, reduce_lr], verbose=1)

# Evaluation
test_loss, test_acc = model.evaluate(val_ds, verbose=0)
print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")
```

```
Epoch 1/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 45ms/step - accuracy: 0.2319 - loss: 5.2
Epoch 2/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.5192 - loss: 1.57
Epoch 3/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.6039 - loss: 1.20
Epoch 4/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.6553 - loss: 1.03
Epoch 5/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.6904 - loss: 0.91
Epoch 6/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7152 - loss: 0.83
Epoch 7/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7316 - loss: 0.79
Epoch 8/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7432 - loss: 0.75
Epoch 9/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7589 - loss: 0.71
Epoch 10/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7685 - loss: 0.67
Epoch 11/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7753 - loss: 0.65
Epoch 12/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7819 - loss: 0.62
Epoch 13/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7900 - loss: 0.60
Epoch 14/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7939 - loss: 0.59
Epoch 15/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7986 - loss: 0.58
Epoch 16/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.8060 - loss: 0.56
Epoch 17/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8118 - loss: 0.54
Epoch 18/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8128 - loss: 0.54
Epoch 19/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8162 - loss: 0.52
Epoch 20/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8210 - loss: 0.52
Epoch 21/30
```

```
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8219 - loss: 0.50
Epoch 22/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8237 - loss: 0.50
Epoch 23/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8300 - loss: 0.49
Epoch 24/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8342 - loss: 0.47
Epoch 25/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8362 - loss: 0.47
Epoch 26/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8335 - loss: 0.47
Epoch 27/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8376 - loss: 0.46
Epoch 28/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8405 - loss: 0.45
Epoch 29/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8418 - loss: 0.45
```

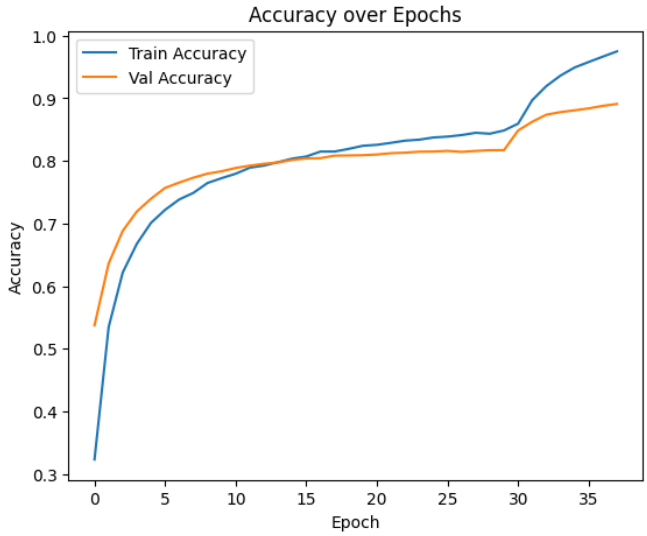
```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras import mixed_precision
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

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

#faster float16
mixed_precision.set_global_policy('mixed_float16')
tf.config.optimizer.set_jit(True)  # XLA Compilation

# Resize and preprocess
x_train_resized = tf.image.resize(x_train, [96, 96])
x_test_resized = tf.image.resize(x_test, [96, 96])
x_train_vgg = preprocess_input(x_train_resized.numpy())
x_test_vgg = preprocess_input(x_test_resized.numpy())

#
train_ds = tf.data.Dataset.from_tensor_slices((x_train_vgg, y_train_cat)).shuffl
val_ds = tf.data.Dataset.from_tensor_slices((x_test_vgg, y_test_cat)).batch(128)

# تحميل VGG16
base_model = VGG16(include_top=False, weights='imagenet', input_shape=(96, 96, 3
base_model.trainable = False
```

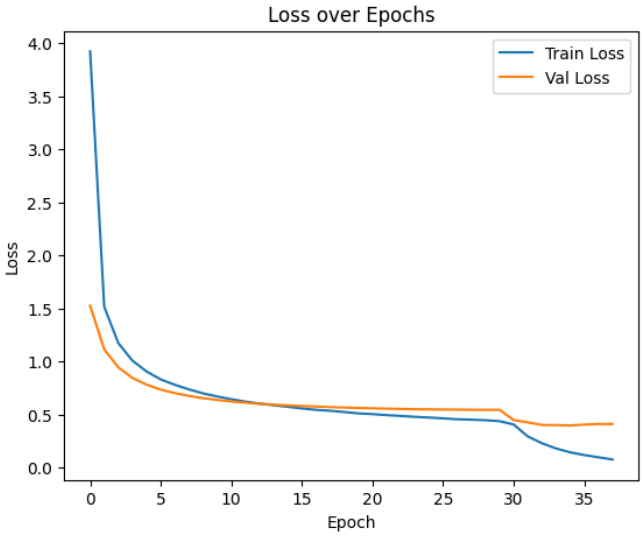
```
base_model.trainable = False

# رأس النموذج
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
output = Dense(10, activation='softmax', dtype='float32')(x)  # لازم float32 في m

model = Model(inputs=base_model.input, outputs=output)

model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbos

# ===== Phase 1: Train head =====
history1 = model.fit(train_ds, validation_data=val_ds, epochs=80,
                     callbacks=[early_stop, reduce_lr], verbose=1)

# ===== Phase 2: Fine-tuning =====
for layer in base_model.layers[-4:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history2 = model.fit(train_ds, validation_data=val_ds, epochs=10,
                     callbacks=[early_stop, reduce_lr], verbose=1)
```

```
Epoch 1/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 16s 32ms/step - accuracy: 0.2228 - loss: 6.266
Epoch 2/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.5066 - loss: 1.6483
Epoch 3/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.6035 - loss: 1.2258
Epoch 4/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.6568 - loss: 1.0403
Epoch 5/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.6949 - loss: 0.9248
Epoch 6/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7183 - loss: 0.8461
Epoch 7/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7375 - loss: 0.7901
```

```
Epoch 8/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.7461 - loss: 0.7483
Epoch 9/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.7643 - loss: 0.7053
Epoch 10/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7697 - loss: 0.6819
Epoch 11/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.7802 - loss: 0.6497
Epoch 12/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7883 - loss: 0.6265
Epoch 13/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7938 - loss: 0.6072
Epoch 14/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.7986 - loss: 0.5941
Epoch 15/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8042 - loss: 0.5770
Epoch 16/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8059 - loss: 0.5614
Epoch 17/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.8141 - loss: 0.5491
Epoch 18/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8145 - loss: 0.5395
Epoch 19/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8206 - loss: 0.5271
Epoch 20/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8225 - loss: 0.5185
Epoch 21/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8262 - loss: 0.5070
Epoch 22/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8281 - loss: 0.4996
Epoch 23/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8316 - loss: 0.4881
Epoch 24/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8331 - loss: 0.4858
Epoch 25/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8360 - loss: 0.4803
Epoch 26/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8398 - loss: 0.4666
Epoch 27/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8426 - loss: 0.4598
Epoch 28/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.8450 - loss: 0.4547
Epoch 29/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.8429 - loss: 0.4522
Epoch 30/30
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 22ms/step - accuracy: 0.8477 - loss: 0.4432
Restoring model weights from the end of the best epoch: 29.
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 18s 36ms/step - accuracy: 0.8527 - loss: 0.426
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.8959 - loss: 0.299
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9168 - loss: 0.236
Epoch 4/10
```

```
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9341 - loss: 0.187
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9471 - loss: 0.149
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9569 - loss: 0.121
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9656 - loss: 0.1017
Epoch 7: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9656 - loss: 0.101
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.9735 - loss: 0.081
Epoch 8: early stopping
Restoring model weights from the end of the best epoch: 5.
Test Accuracy: 0.8809, Test Loss: 0.3976
```





```
for layer in model.layers:
    print(layer.name)
```

```
input_layer
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_pool
block4_conv1
block4_conv2
block4_conv3
block4_pool
block5_conv1
block5_conv2
block5_conv3
block5_pool
global_average_pooling2d
dense
dropout
dense_1
```

```
model.save("vgg_light_finetuned22.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` o
```

model summary

```
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 96, 96, 3)	0
cast (Cast)	(None, 96, 96, 3)	0
block1_conv1 (Conv2D)	(None, 96, 96, 64)	1,792
block1_conv2 (Conv2D)	(None, 96, 96, 64)	36,928
block1_pool (MaxPooling2D)	(None, 48, 48, 64)	0

block2_conv1 (Conv2D)	(None, 48, 48, 128)	73,856
block2_conv2 (Conv2D)	(None, 48, 48, 128)	147,584
block2_pool (MaxPooling2D)	(None, 24, 24, 128)	0
block3_conv1 (Conv2D)	(None, 24, 24, 256)	295,168
block3_conv2 (Conv2D)	(None, 24, 24, 256)	590,080
block3_conv3 (Conv2D)	(None, 24, 24, 256)	590,080
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0
block4_conv1 (Conv2D)	(None, 12, 12, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 12, 12, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 12, 12, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 6, 6, 512)	0
block5_conv1 (Conv2D)	(None, 6, 6, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 6, 6, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 6, 6, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 3, 3, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 64)	32,832
dropout (Dropout)	(None, 64)	0
cast_1 (Cast)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Total params: 28,973,988 (110.53 MB)
Trainable params: 7,112,906 (27.13 MB)
Non-trainable params: 7,635,264 (29.13 MB)
Optimizer params: 14,225,818 (54.27 MB)

Evaluate the model on test set

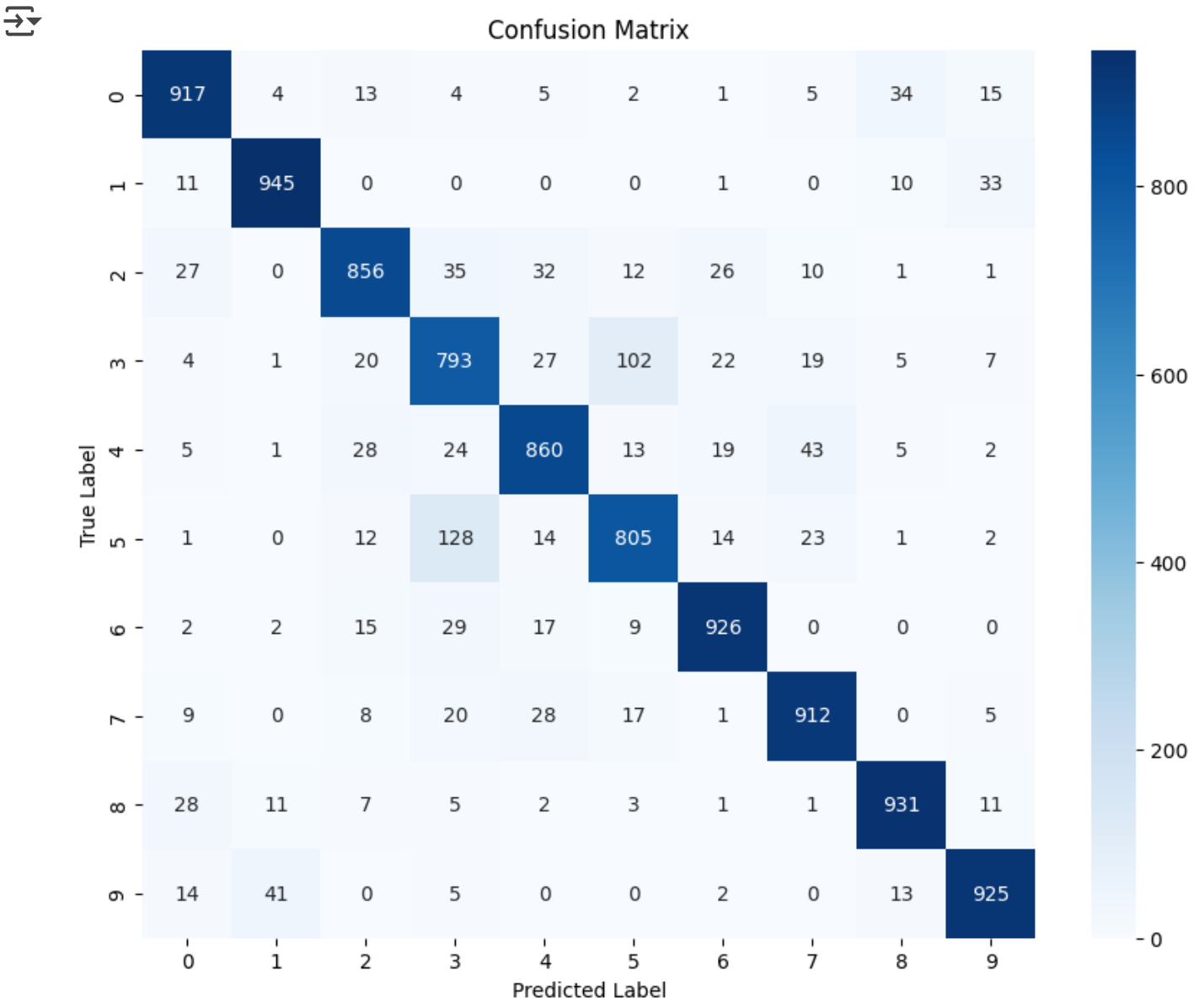
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# توقعات النموذج
y_pred_probs = model.predict(val_ds)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test_cat, axis=1)

# Classification Report
print("Classification Report:\n", classification_report(y_true, y_pred))
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      1000
           1       0.94      0.94      0.94      1000
           2       0.89      0.86      0.87      1000
           3       0.76      0.79      0.78      1000
           4       0.87      0.86      0.87      1000
           5       0.84      0.81      0.82      1000
           6       0.91      0.93      0.92      1000
           7       0.90      0.91      0.91      1000
           8       0.93      0.93      0.93      1000
           9       0.92      0.93      0.92      1000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

```
Start coding or generate with AI.
```

Confusion Matrix

```
conf_mat = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

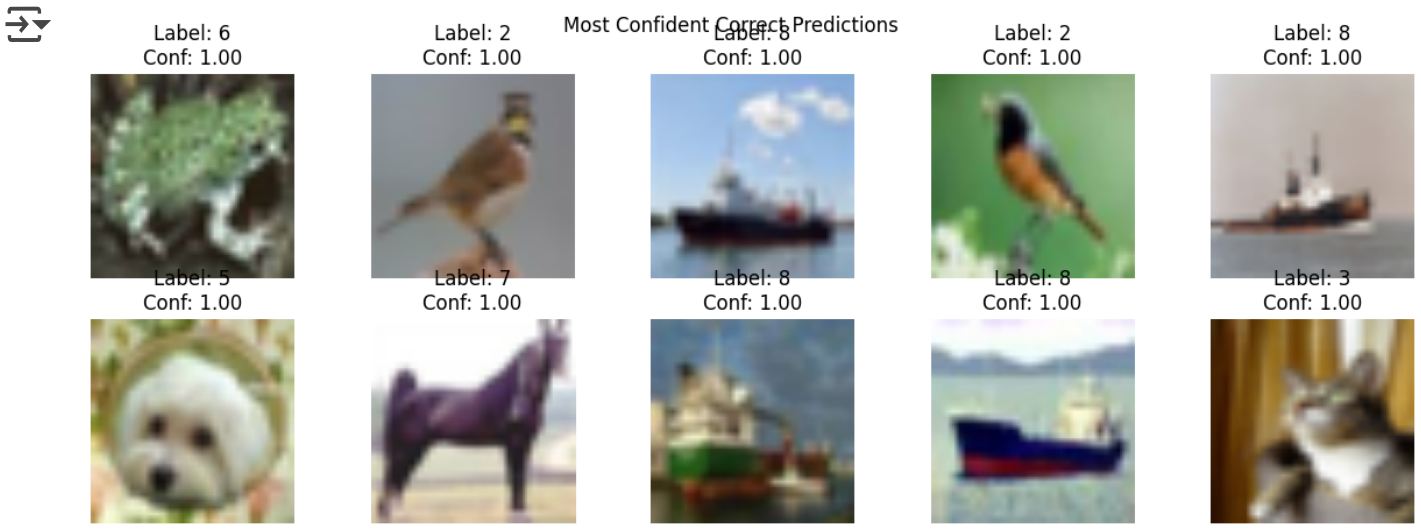


High Confidence Errors

```
wrong_indices = np.where(y_pred != y_true)[0]
wrong_confidences = np.max(y_pred_probs[wrong_indices], axis=1)
top_wrong = wrong_indices[np.argsort(wrong_confidences)[-10:]]


plt.figure(figsize=(15, 5))
for i, idx in enumerate(top_wrong):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test_resized[idx].numpy().astype("uint8"))
    plt.title(f"True: {y_true[idx]}, Pred: {y_pred[idx]}\nConf: {wrong_confider
    plt.axis("off")
plt.suptitle("Most Confident Wrong Predictions")
plt.show()
```

Top Confident Correct Predictions

```
correct_indices = np.where(y_pred == y_true)[0]
correct_confidences = np.max(y_pred_probs[correct_indices], axis=1)
top_correct = correct_indices[np.argsort(correct_confidences)[-10:]]

plt.figure(figsize=(15, 5))
for i, idx in enumerate(top_correct):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test_resized[idx].numpy().astype("uint8"))
    plt.title(f"Label: {y_true[idx]}\nConf: {correct_confidences[np.argsort(cor
    plt.axis("off")
plt.suptitle("Most Confident Correct Predictions")
plt.show()
```

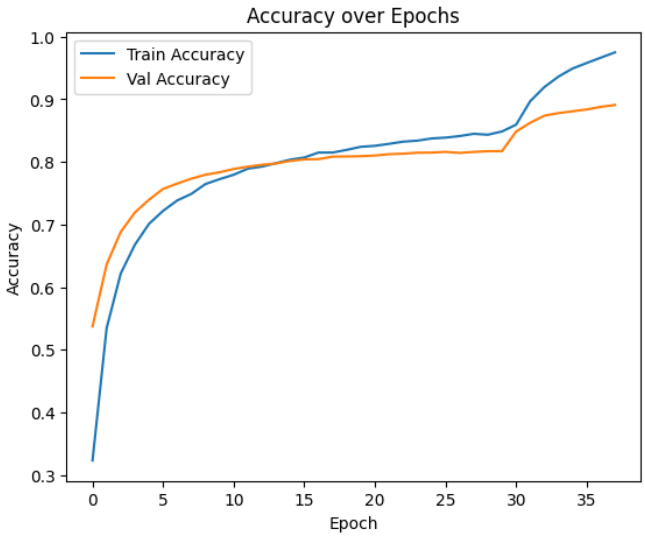


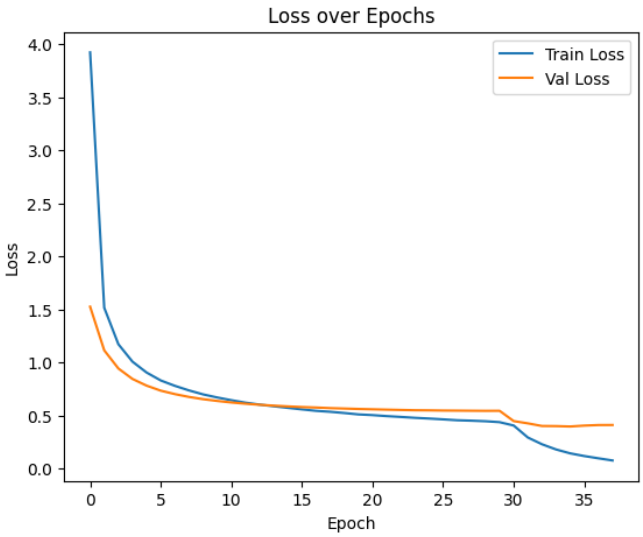
Accuracy & Loss Curves

```
test_loss, test_acc = model.evaluate(val_ds, verbose=0)
```

```
print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")

def merge_histories(h1, h2):
    merged = {}
    for key in h1.history:
        merged[key] = h1.history[key] + h2.history[key]
    return merged

def plot_training_curves(history):
    acc = history['accuracy']
    val_acc = history['val_accuracy']
    loss = history['loss']
    val_loss = history['val_loss']

    plt.figure(figsize=(14, 5))

    # Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(acc, label='Train Accuracy')
    plt.plot(val_acc, label='Val Accuracy')
    plt.title('Accuracy over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(loss, label='Train Loss')
    plt.plot(val_loss, label='Val Loss')
    plt.title('Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()


merged_history = merge_histories(history1, history2)
plot_training_curves(merged_history)
```

```
Test Accuracy: 0.8809, Test Loss: 0.3976
```





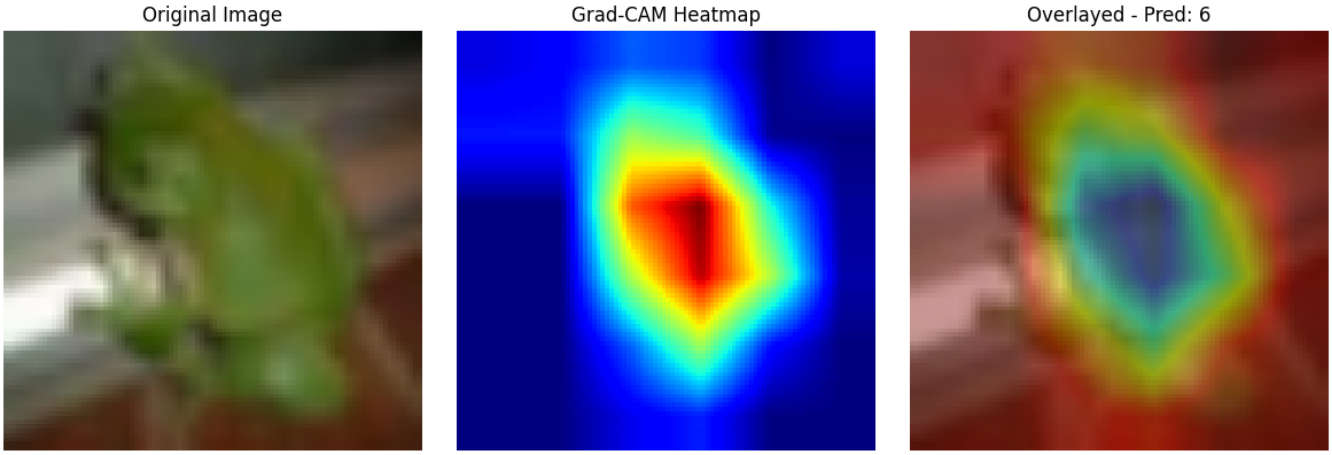
Grad-CAM on VGG model

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2


index = 7
img_input = x_test_vgg[index]
original_img = x_test_resized[index].numpy()


img_tensor = tf.convert_to_tensor(np.expand_dims(img_input, axis=0))


last_conv_layer_name = "block5_conv3"
```

```
grad_model = tf.keras.models.Model(
    [model.inputs],
    [model.get_layer(last_conv_layer_name).output, model.output]
)


with tf.GradientTape() as tape:
    conv_outputs, predictions = grad_model(img_tensor)
    predicted_class = tf.argmax(predictions[0])
    loss = predictions[:, predicted_class]

grads = tape.gradient(loss, conv_outputs)
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

conv_outputs = conv_outputs[0]
heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
heatmap = tf.squeeze(heatmap)


heatmap = tf.squeeze(heatmap)


if len(heatmap.shape) != 2 or tf.reduce_max(heatmap).numpy() == 0:
    heatmap = tf.ones((12, 12))


heatmap = np.maximum(heatmap, 0)
heatmap = heatmap / np.max(heatmap)


heatmap = np.uint8(255 * heatmap)
heatmap = cv2.resize(heatmap, (96, 96))
heatmap_color = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)


original_img_norm = (original_img - original_img.min()) / (original_img.max() -
original_img_uint8 = np.uint8(original_img_norm * 255)


alpha = 0.4
overlay = cv2.addWeighted(original_img_uint8, 1 - alpha, heatmap_color, alpha,


plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
```

```
plt.title("Original Image")
plt.imshow(original_img_uint8.astype('uint8'))
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Grad-CAM Heatmap")
plt.imshow(heatmap, cmap='jet')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title(f"Overlayed - Pred: {predicted_class.numpy()}")
plt.imshow(overlay)
plt.axis('off')

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237:
Expected: [['keras_tensor_25']]
Received: inputs=Tensor(shape=(1, 96, 96, 3))
  raise ValueError(msg)
```



The Bonus Autoencoder + KMeans + visualization

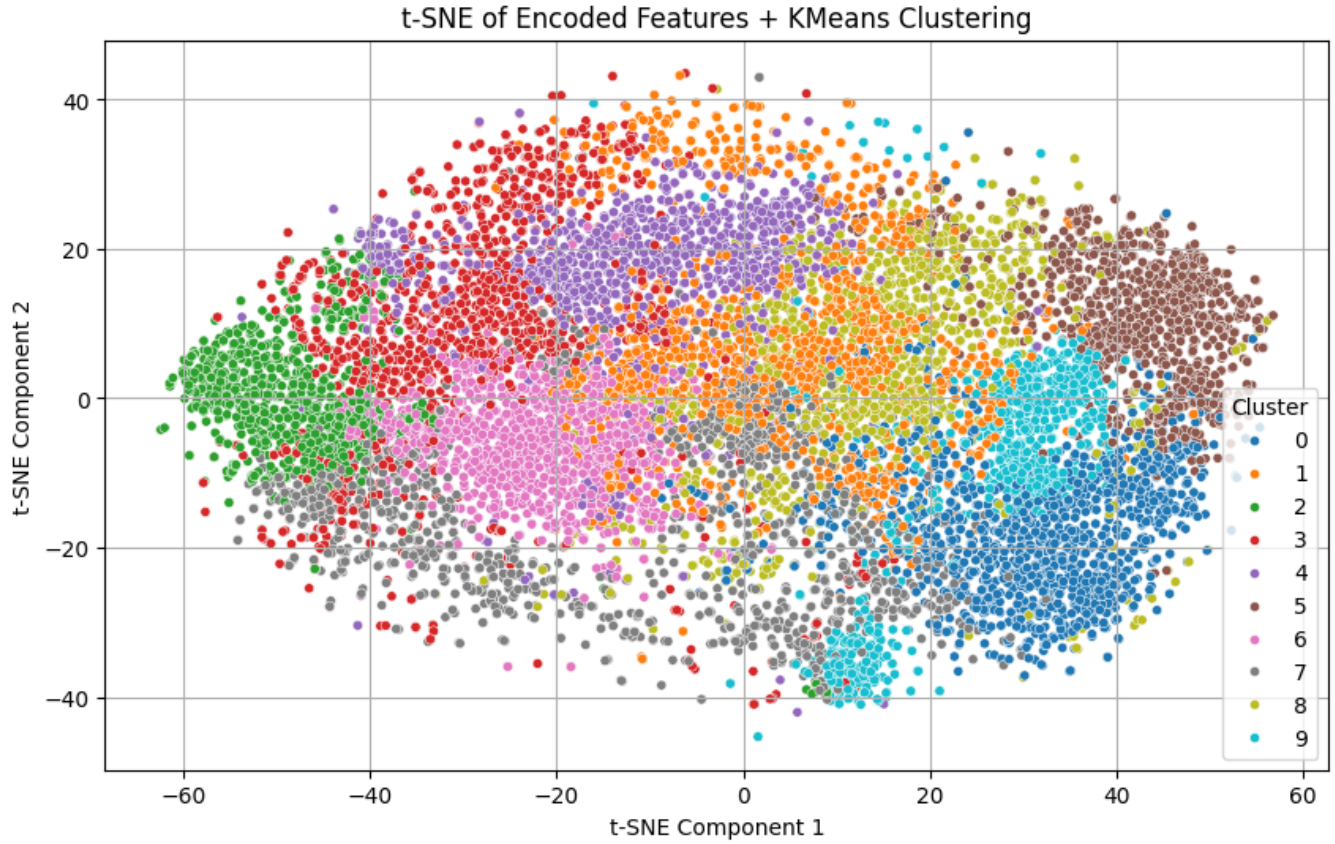
```
!pip install tensorflow --quiet
```

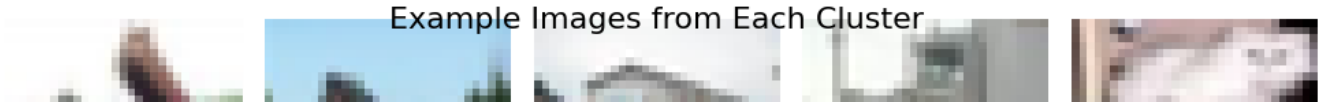
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.metrics import accuracy_score, adjusted_rand_score
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.optimizers import Adam
import seaborn as sns

# ===== Step 1: Load and normalize CIFAR-10 =====
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_data = np.concatenate((x_train, x_test), axis=0).astype('float32') / 255.0
y_data = np.concatenate((y_train, y_test), axis=0).flatten()

# ===== Step 2: Build Autoencoder =====
input_img = Input(shape=(32, 32, 3))
x = Conv2D(32, 3, activation='relu', padding='same')(input_img)
x = MaxPooling2D(2, padding='same')(x)
x = Conv2D(64, 3, activation='relu', padding='same')(x)
encoded = MaxPooling2D(2, padding='same')(x)  # (8, 8, 64)

x = Conv2D(64, 3, activation='relu', padding='same')(encoded)
x = UpSampling2D(2)(x)
x = Conv2D(32, 3, activation='relu', padding='same')(x)
x = UpSampling2D(2)(x)
decoded = Conv2D(3, 3, activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(1e-3), loss='mse')

# ===== Step 3: Train Autoencoder =====
autoencoder.fit(x_data[:40000], x_data[:40000],
                epochs=15, batch_size=256,
                validation_split=0.1, shuffle=True)

# ===== Step 4: Encode Data =====
encoder = Model(input_img, encoded)
features = encoder.predict(x_data[:10000])
```

```
features_flat = features.reshape((features.shape[0], -1))
true_labels = y_data[:10000]

# ===== Step 5: Clustering with KMeans =====
kmeans = KMeans(n_clusters=10, random_state=42)
pred_clusters = kmeans.fit_predict(features_flat)

# ===== Step 6: Evaluate clustering (Unsupervised Accuracy) =====
# ملاحظة: في التعلم غير الخاضع للإشراف لا يوجد "accuracy" مباشرة
# لكن نستخدم adjusted_rand_score كمقياس رسمي
ari_score = adjusted_rand_score(true_labels, pred_clusters)
print(f"Adjusted Rand Index (ARI): {ari_score:.4f}")

# ===== Step 7: Visualize Clusters with t-SNE =====
tsne = TSNE(n_components=2, random_state=42)
tsne_result = tsne.fit_transform(features_flat)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=tsne_result[:, 0], y=tsne_result[:, 1], hue=pred_clusters, pa
plt.title("t-SNE of Encoded Features + KMeans Clustering")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.grid(True)
plt.legend(title='Cluster')
plt.show()

# ===== Step 8: Show example images from each cluster =====
def show_cluster_examples(images, labels, cluster_ids, samples_per_cluster=5):
    plt.figure(figsize=(samples_per_cluster*2, 20))
    for cluster in range(10):
        idxs = np.where(cluster_ids == cluster)[0][:samples_per_cluster]
        for i, idx in enumerate(idxs):
            plt.subplot(10, samples_per_cluster, cluster*samples_per_cluster +
            plt.imshow(images[idx])
            plt.axis('off')
            if i == 0:
                plt.ylabel(f"Cluster {cluster}", fontsize=12)
    plt.suptitle("Example Images from Each Cluster", fontsize=16)
    plt.tight_layout()
    plt.show()

show_cluster_examples(x_data[:10000], true_labels, pred_clusters)
```

```
Epoch 1/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 7s 26ms/step - loss: 0.0322 - val_loss: 0.0083
Epoch 2/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0077 - val_loss: 0.0064
Epoch 3/15
```

```
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0062 - val_loss: 0.0056
Epoch 4/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0055 - val_loss: 0.0051
Epoch 5/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0050 - val_loss: 0.0047
Epoch 6/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0046 - val_loss: 0.0044
Epoch 7/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0044 - val_loss: 0.0045
Epoch 8/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0042 - val_loss: 0.0041
Epoch 9/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0041 - val_loss: 0.0039
Epoch 10/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0040 - val_loss: 0.0042
Epoch 11/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0038 - val_loss: 0.0037
Epoch 12/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0038 - val_loss: 0.0036
Epoch 13/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0036 - val_loss: 0.0037
Epoch 14/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0036 - val_loss: 0.0035
Epoch 15/15
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0035 - val_loss: 0.0035
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Adjusted Rand Index (ARI): 0.0511
```



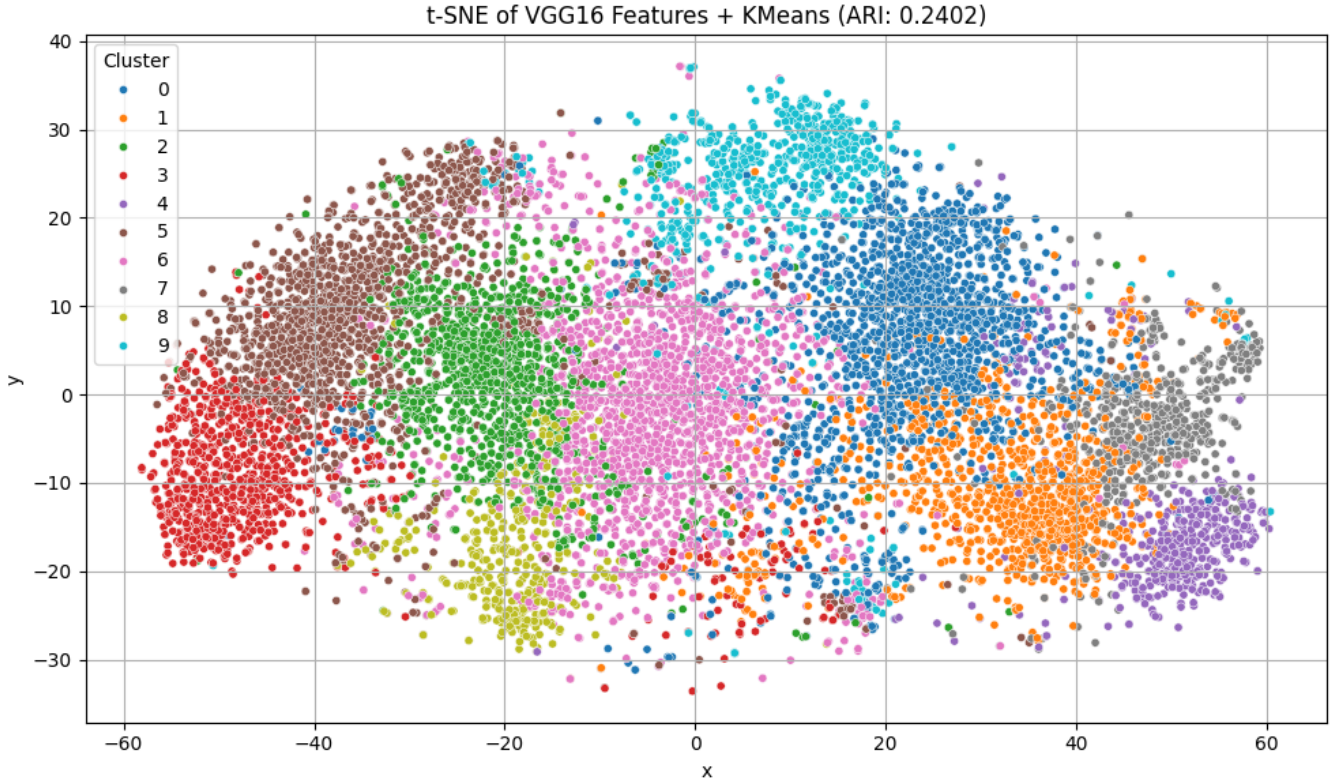


```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.metrics import adjusted_rand_score
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D
import tensorflow as tf

# Step 1: Load CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_data = np.concatenate([x_train, x_test])
y_data = np.concatenate([y_train, y_test]).flatten()

# Step 2: Use 10,000 samples for better clustering
x_subset = x_data[:10000]
y_subset = y_data[:10000]

# Resize and preprocess
x_resized = tf.image.resize(x_subset, [224, 224]).numpy()
x_preprocessed = preprocess_input(x_resized)

# Step 3: VGG16 model + GlobalAveragePooling for better feature reduction
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = vgg.output
x = GlobalAveragePooling2D()(x)
feature_model = Model(inputs=vgg.input, outputs=x)

# Step 4: Extract features
features = feature_model.predict(x_preprocessed, batch_size=64, verbose=1)

# Step 5: KMeans clustering
kmeans = KMeans(n_clusters=10, random_state=42)
clusters = kmeans.fit_predict(features)

# Step 6: Evaluate clustering
ari = adjusted_rand_score(y_subset, clusters)
print(f"\n🎯 Adjusted Rand Index (ARI): {ari:.4f}")

# Step 7: Visualize with t-SNE
tsne = TSNE(n_components=2, perplexity=50, random_state=42)
tsne_result = tsne.fit_transform(features)
```

```
# Plot
df = pd.DataFrame({
    'x': tsne_result[:, 0],
    'y': tsne_result[:, 1],
    'cluster': clusters
})

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='x', y='y', hue='cluster', palette='tab10', s=20)
plt.title(f"t-SNE of VGG16 Features + KMeans (ARI: {ari:.4f})")
plt.grid(True)
plt.legend(title='Cluster')
plt.tight_layout()
plt.show()
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 48s 181ms/step
```

```
🎯 Adjusted Rand Index (ARI): 0.2402
```



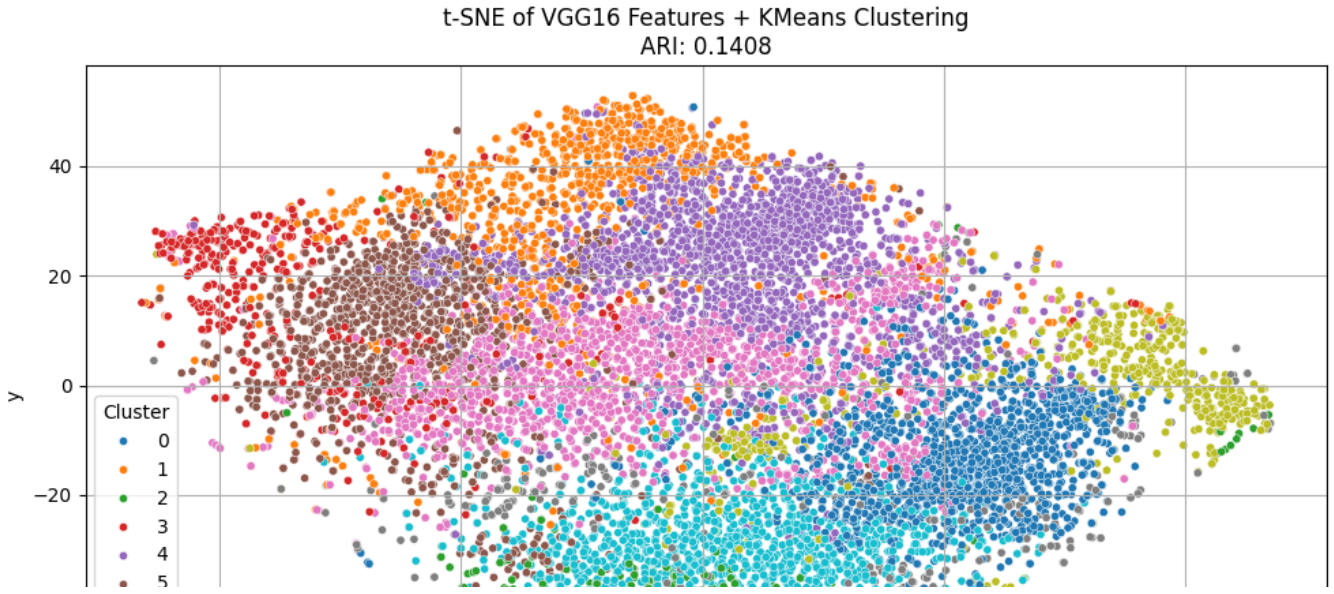

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.metrics import adjusted_rand_score
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
import tensorflow as tf
```

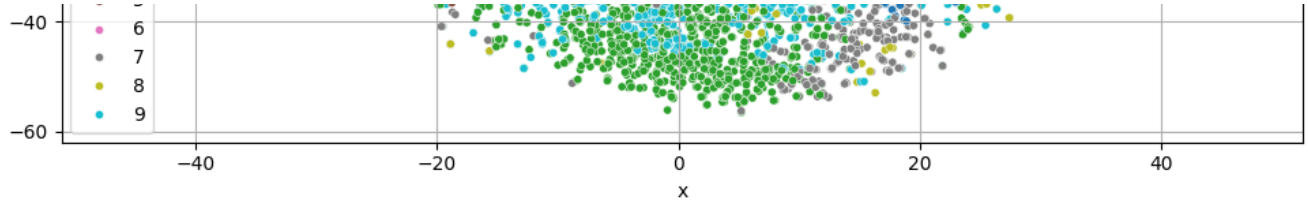
```
# Step 1: Load CIFAR-10 and preprocess
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_data = np.concatenate([x_train, x_test])
y_data = np.concatenate([y_train, y_test]).flatten()

x_subset = x_data[:10000]
y_subset = y_data[:10000]

x_resized = tf.image.resize(x_subset, [224, 224]).numpy()
x_preprocessed = preprocess_input(x_resized)

# Step 2: VGG16 feature model with enhancements
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.4)(x)
x = Dense(128, activation='relu')(x)
feature_output = Dropout(0.4)(x)

feature_model = Model(inputs=base_model.input, outputs=feature_output)

# Step 3: Feature extraction
features = feature_model.predict(x_preprocessed, batch_size=64, verbose=1)

# Step 4: Clustering
kmeans = KMeans(n_clusters=10, random_state=42)
clusters = kmeans.fit_predict(features)

# Step 5: Evaluate with ARI
ari = adjusted_rand_score(y_subset, clusters)
print(f"\n🎯 Adjusted Rand Index (ARI): {ari:.4f}")

# Step 6: Save CSV
df_results = pd.DataFrame({
    'Image_Index': np.arange(len(y_subset)),
    'True_Label': y_subset,
    'Cluster': clusters
})
df_results.to_csv("vgg_clusters.csv", index=False)

# Step 7: t-SNE plot
tsne_result = TSNE(n_components=2, perplexity=50, random_state=42).fit_transfor
df_tsne = pd.DataFrame({
```

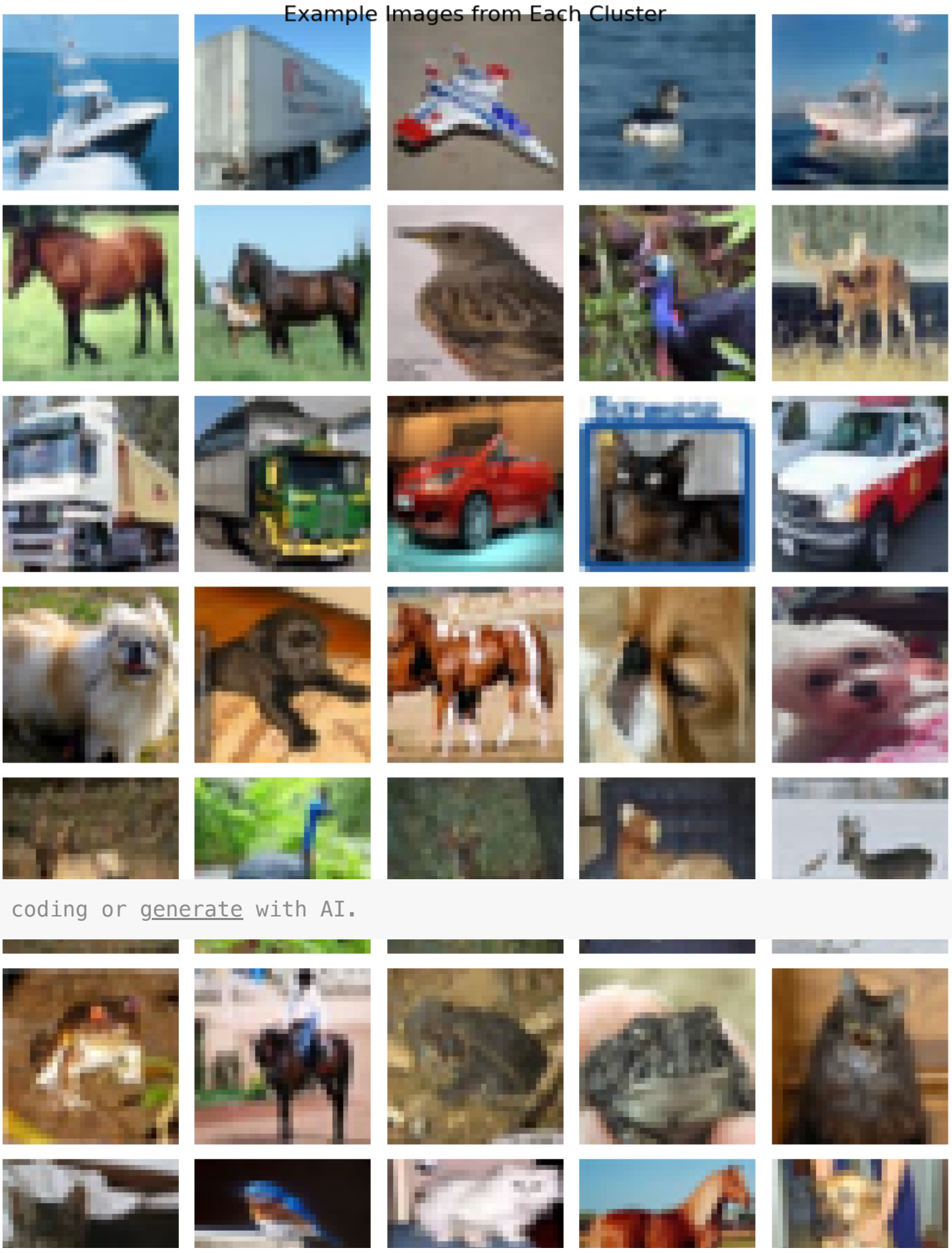
```
    'x': tsne_result[:, 0],
    'y': tsne_result[:, 1],
    'cluster': clusters
})

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_tsne, x='x', y='y', hue='cluster', palette='tab10', s=2
plt.title(f"t-SNE of VGG16 Features + KMeans Clustering\nARI: {ari:.4f}")
plt.grid(True)
plt.tight_layout()
plt.legend(title="Cluster")
plt.show()

# Step 8: Show example images from each cluster
def show_cluster_examples(images, cluster_ids, samples_per_cluster=5):
    plt.figure(figsize=(samples_per_cluster * 2, 20))
    for cluster in range(10):
        idxs = np.where(cluster_ids == cluster)[0][:samples_per_cluster]
        for i, idx in enumerate(idxs):
            plt.subplot(10, samples_per_cluster, cluster * samples_per_cluster
            plt.imshow(images[idx])
            plt.axis('off')
            if i == 0:
                plt.ylabel(f"Cluster {cluster}", fontsize=12)
    plt.suptitle("Example Images from Each Cluster", fontsize=16)
    plt.tight_layout()
    plt.show()

show_cluster_examples(x_subset, clusters)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 25s 154ms/step
```

```
🎯 Adjusted Rand Index (ARI): 0.1408
```







Start coding or generate with AI.