ANN Model

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

```
(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

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
Training data shape: (50000, 32, 32, 3)
Test data shape: (10000, 32, 32, 3)
```

```
model = models.Sequential()


model.add(layers.Flatten(input_shape=(32, 32, 3)))


model.add(layers.Dense(512, activation='relu'))


model.add(layers.Dense(256, activation='relu'))


model.add(layers.Dense(128, activation='relu'))


model.add(layers.Dense(10, activation='softmax'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential m
  super().__init__(**kwargs)
```

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

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 512)	1,573,376
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 128)	32,896
dense_3 (Dense)	(None, 10)	1,290

Total params: 1,738,890 (6.63 MB)
Trainable params: 1,738,890 (6.63 MB)
Non-trainable params: 0 (0.00 B)

CNN Model

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

```
(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


y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)



print("Training shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

```
Training shape: (50000, 32, 32, 3)
Test shape: (10000, 32, 32, 3)
```

```
model = models.Sequential()

model.add(layers.Conv2D(32, (3,3), activation='relu',
                        padding='same',
                        input_shape=(32,32,3)))
model.add(layers.MaxPooling2D((2,2)))
```

```
model.add(layers.Conv2D(64, (5,5), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2,2)))


model.add(layers.Conv2D(128, (7,7), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2,2)))


model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))


model.add(layers.Dense(10, activation='softmax'))
```

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

```
model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_10 (Conv2D)	(None, 16, 16, 64)	51,264
max_pooling2d_10 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_11 (Conv2D)	(None, 8, 8, 128)	401,536
max_pooling2d_11 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_4 (Flatten)	(None, 2048)	0
dense_10 (Dense)	(None, 128)	262,272
dropout_3 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 10)	1,290

Total params: 717,258 (2.74 MB)
Trainable params: 717,258 (2.74 MB)
Non-trainable params: 0 (0.00 B)

TRAIN MODEL ANN

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import time
```

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


x_train = x_train / 255.0
x_test = x_test / 255.0


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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 14s 0us/step
```

```
ann_model = Sequential([
    Flatten(input_shape=(32, 32, 3)),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential m
  super().__init__(**kwargs)
```

```
ann_start_time = time.time()

ann_history = ann_model.fit(
    x_train, y_train,
    epochs=10,
    batch_size=64,
    validation_data=(x_test, y_test),
    verbose=1
)

ann_training_time = time.time() - ann_start_time


ann_model.save_weights("ann_weights.weights.h5")
```

```
print("ANN Training Time:", ann_training_time, "seconds")
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 24s 29ms/step - accuracy: 0.2782 - loss: 1.9934 - val_accuracy: 0.3657 - val_loss: 1.7378
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.3892 - loss: 1.6869 - val_accuracy: 0.3979 - val_loss: 1.6689
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 43s 31ms/step - accuracy: 0.4233 - loss: 1.6027 - val_accuracy: 0.4490 - val_loss: 1.5474
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 38s 27ms/step - accuracy: 0.4453 - loss: 1.5444 - val_accuracy: 0.4481 - val_loss: 1.5305
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - accuracy: 0.4647 - loss: 1.4904 - val_accuracy: 0.4544 - val_loss: 1.5145
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - accuracy: 0.4815 - loss: 1.4550 - val_accuracy: 0.4594 - val_loss: 1.5134
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.4886 - loss: 1.4265 - val_accuracy: 0.4709 - val_loss: 1.4959
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 29ms/step - accuracy: 0.4990 - loss: 1.3936 - val_accuracy: 0.4859 - val_loss: 1.4493
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.5051 - loss: 1.3682 - val_accuracy: 0.4803 - val_loss: 1.4887
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.5164 - loss: 1.3518 - val_accuracy: 0.4806 - val_loss: 1.4374
ANN Training Time: 258.47081780433655 seconds
```

TRAIN MODEL CNN

```
cnn_model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3), padding='same'),
    MaxPooling2D((2,2)),

    Conv2D(64, (5,5), activation='relu', padding='same'),
    MaxPooling2D((2,2)),

    Conv2D(128, (7,7), activation='relu', padding='same'),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seque
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
cnn_start_time = time.time()

cnn_history = cnn_model.fit(
    x_train, y_train,
    epochs=10,
    batch_size=64,
    validation_data=(x_test, y_test),
    verbose=1
)

cnn_training_time = time.time() - cnn_start_time


cnn_model.save_weights("cnn_weights.weights.h5")

print("CNN Training Time:", cnn_training_time, "seconds")
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 309s 393ms/step - accuracy: 0.2941 - loss: 1.8851 - val_accuracy: 0.5368 - val_loss: 1.2695
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 302s 386ms/step - accuracy: 0.5599 - loss: 1.2443 - val_accuracy: 0.6265 - val_loss: 1.0583
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 306s 391ms/step - accuracy: 0.6472 - loss: 1.0167 - val_accuracy: 0.6780 - val_loss: 0.9446
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 308s 394ms/step - accuracy: 0.6990 - loss: 0.8722 - val_accuracy: 0.7031 - val_loss: 0.8477
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 322s 395ms/step - accuracy: 0.7354 - loss: 0.7684 - val_accuracy: 0.7112 - val_loss: 0.8405
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 322s 394ms/step - accuracy: 0.7679 - loss: 0.6791 - val_accuracy: 0.7265 - val_loss: 0.8204
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 318s 389ms/step - accuracy: 0.7925 - loss: 0.5913 - val_accuracy: 0.7505 - val_loss: 0.7786
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 310s 396ms/step - accuracy: 0.8169 - loss: 0.5239 - val_accuracy: 0.7408 - val_loss: 0.7965
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 321s 395ms/step - accuracy: 0.8386 - loss: 0.4594 - val_accuracy: 0.7429 - val_loss: 0.8279
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 313s 400ms/step - accuracy: 0.8602 - loss: 0.3949 - val_accuracy: 0.7519 - val_loss: 0.8072
CNN Training Time: 3132.262302160263 seconds
```

MODEL ANN AND CNN

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

```
ann_test_loss, ann_test_accuracy = ann_model.evaluate(x_test, y_test, verbose=0)

print("ANN Test Accuracy:", ann_test_accuracy)
print("ANN Test Loss:", ann_test_loss)
```

```
ANN Test Accuracy: 0.4805999994277954
ANN Test Loss: 1.4374479055404663
```

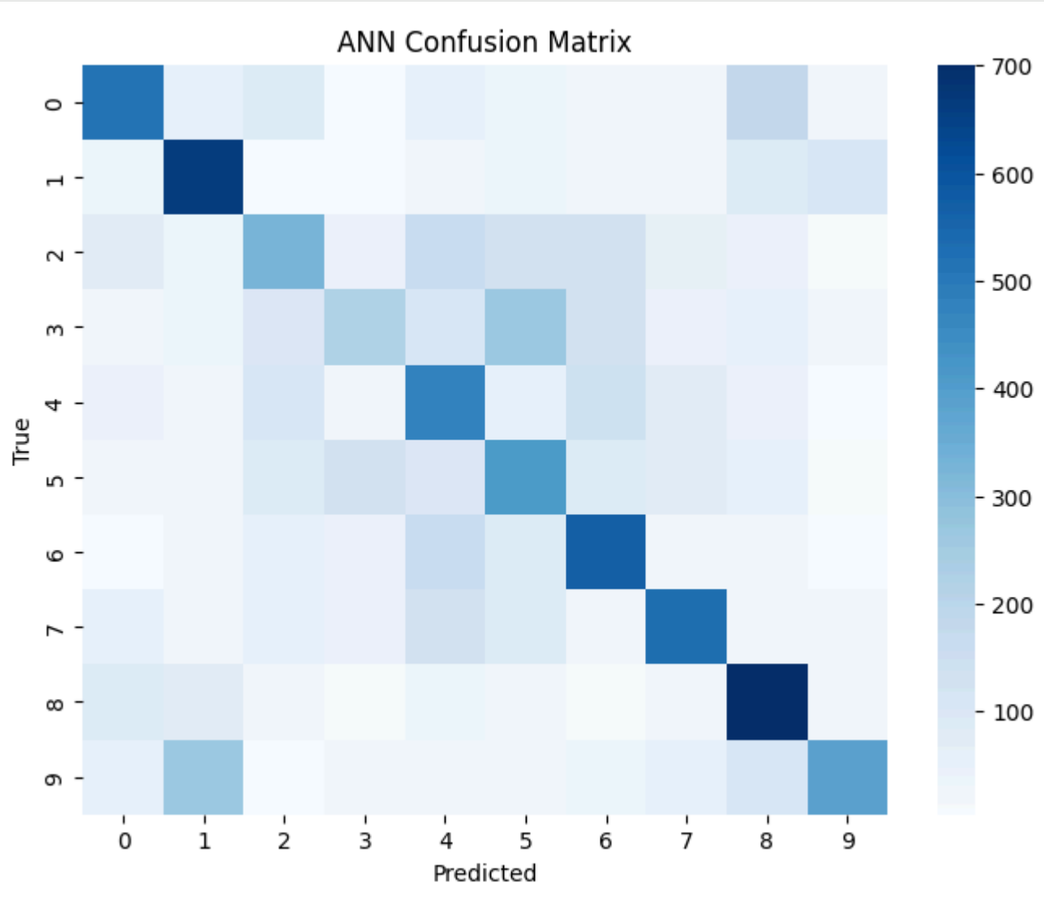
```
test_loss, test_accuracy = cnn_model.evaluate(x_test, y_test, verbose=0)

print("CNN Test Accuracy:", test_accuracy)
print("CNN Test Loss:", test_loss)
```

```
CNN Test Accuracy: 0.7519000172615051
CNN Test Loss: 0.807242214679718
```

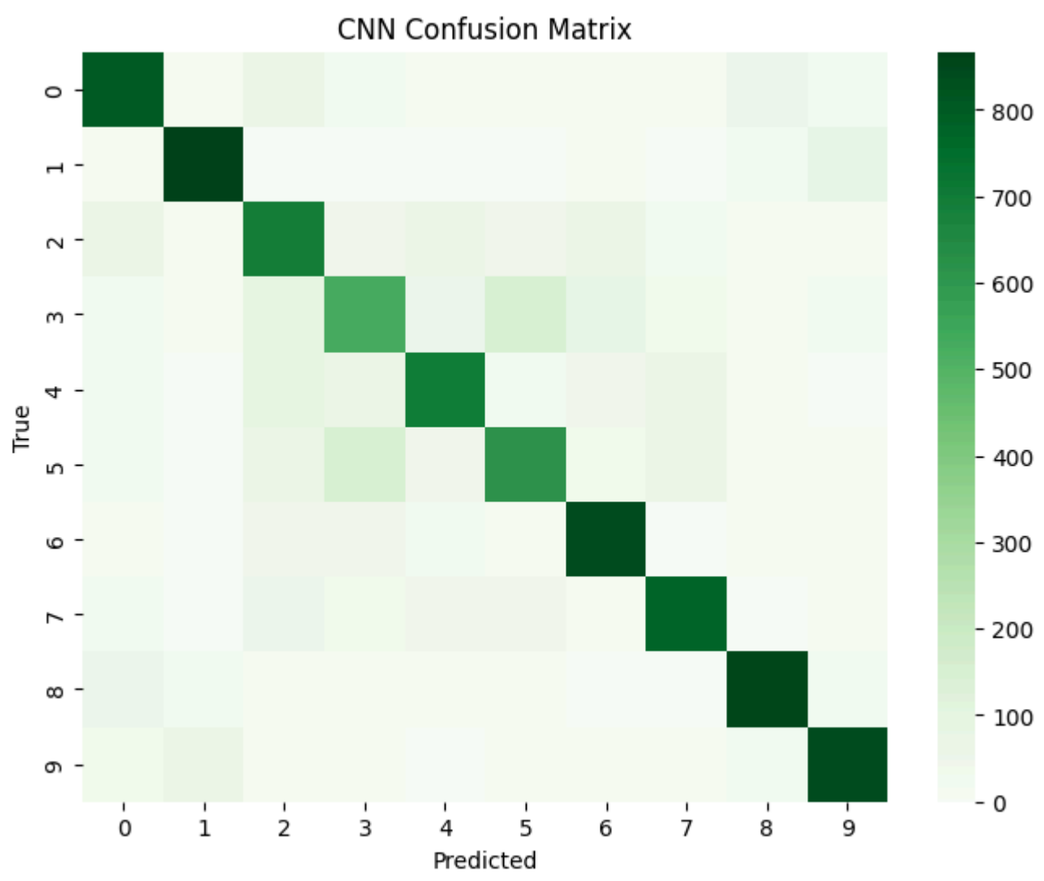
```
ann_predictions = ann_model.predict(x_test)
ann_pred_classes = np.argmax(ann_predictions, axis=1)


cnn_predictions = cnn_model.predict(x_test)
cnn_pred_classes = np.argmax(cnn_predictions, axis=1)


true_classes = np.argmax(y_test, axis=1)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 17s 53ms/step
```

```
ann_cm = confusion_matrix(true_classes, ann_pred_classes)

plt.figure(figsize=(8,6))
sns.heatmap(ann_cm, annot=False, cmap="Blues")
plt.title("ANN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```




```
cnn_cm = confusion_matrix(true_classes, cnn_pred_classes)

plt.figure(figsize=(8,6))
sns.heatmap(cnn_cm, annot=False, cmap="Greens")
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```




```
print("ANN Classification Report:\n")
print(classification_report(
    true_classes,
    ann_pred_classes,
    target_names=class_names
))
```

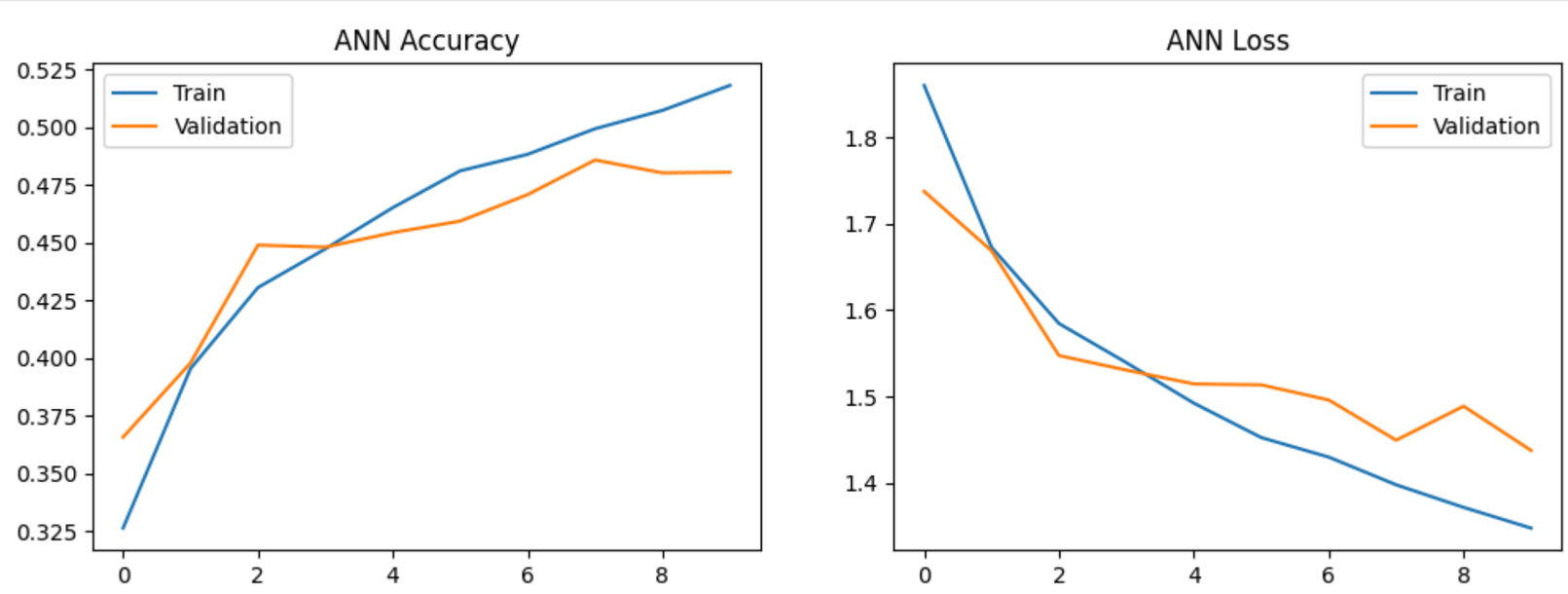
```
ANN Classification Report:

              precision    recall  f1-score   support

    Airplane       0.57      0.52      0.54      1000
  Automobile       0.55      0.66      0.60      1000
```

```
              Bird       0.38      0.32      0.35      1000
               Cat       0.40      0.22      0.29      1000
              Deer       0.38      0.48      0.42      1000
               Dog       0.35      0.41      0.38      1000
              Frog       0.49      0.57      0.52      1000
             Horse       0.57      0.53      0.55      1000
              Ship       0.54      0.70      0.61      1000
             Truck       0.61      0.39      0.48      1000

          accuracy                           0.48     10000
         macro avg       0.48      0.48      0.47     10000
      weighted avg       0.48      0.48      0.47     10000
```

```
print("CNN Classification Report:\n")
print(classification_report(
    true_classes,
    cnn_pred_classes,
    target_names=class_names
))
```

```
CNN Classification Report:

              precision    recall  f1-score   support

    Airplane       0.75      0.80      0.77      1000
  Automobile       0.88      0.87      0.87      1000
        Bird       0.62      0.69      0.65      1000
         Cat       0.59      0.53      0.56      1000
        Deer       0.74      0.70      0.72      1000
         Dog       0.68      0.62      0.65      1000
        Frog       0.77      0.84      0.81      1000
       Horse       0.81      0.78      0.79      1000
        Ship       0.85      0.85      0.85      1000
       Truck       0.82      0.84      0.83      1000

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000
```

```
def plot_history(history, title):
    plt.figure(figsize=(12,4))

    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='Train')
    plt.plot(history.history['val_accuracy'], label='Validation')
    plt.title(f'{title} Accuracy')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train')
    plt.plot(history.history['val_loss'], label='Validation')
    plt.title(f'{title} Loss')
    plt.legend()

    plt.show()

plot_history(ann_history, "ANN")
plot_history(cnn_history, "CNN")
```




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

```
misclassified = np.where(cnn_pred_classes != true_classes)[0][:10]

plt.figure(figsize=(15,6))

for i, index in enumerate(misclassified):
    plt.subplot(2,5,i+1)
```

```
        plt.imshow(x_test[index])
        plt.title(
            f"True: {class_names[true_classes[index]]}\n"
            f"Pred: {class_names[cnn_pred_classes[index]]}"
        )
        plt.axis('off')

plt.show()
```

Summary:-

This assignment presents a comparative study of ANN and CNN for image classification using the CIFAR-10 dataset. Both models are trained and evaluated on accuracy, loss, and classification metrics. The results clearly show that CNN performs better than ANN by effectively learning spatial features from images. Thus, CNN proves to be a more powerful and reliable model for image-based tasks.