**Neural Networks & Deep Learning**

**Quiz**

**Name: Madhu Kumar Sunkaraboina**

**#700s: 700745875**

**GitHub Link:** [**https://github.com/mxs58750/Quiz.git**](https://github.com/mxs58750/Quiz.git)

import tensorflow as tf

from tensorflow.keras.datasets import cifar100

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Load CIFAR-100 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar100.load\_data()

# Normalize data

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Convert class vectors to binary class matrices

y\_train = tf.keras.utils.to\_categorical(y\_train, 100)

y\_test = tf.keras.utils.to\_categorical(y\_test, 100)

# Build a simple CNN model

def create\_model():

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

MaxPooling2D((2, 2)),

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

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(512, activation='relu'),

Dropout(0.5),

Dense(100, activation='softmax')

])

return model

model = create\_model()

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the CNN model

history = model.fit(x\_train, y\_train, epochs=25, batch\_size=64, validation\_data=(x\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc}')

# Use Grid Search to optimize hyperparameters

!pip install keras-tuner

from kerastuner.tuners import RandomSearch

def build\_model(hp):

model = Sequential()

model.add(Conv2D(hp.Int('conv\_1', min\_value=32, max\_value=128, step=32), (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D((2, 2)))

for i in range(hp.Int('num\_conv\_layers', 1, 3)):

model.add(Conv2D(hp.Int(f'conv\_{i+2}', min\_value=32, max\_value=128, step=32), (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(hp.Int('dense\_units', min\_value=128, max\_value=512, step=128), activation='relu'))

model.add(Dropout(hp.Float('dropout\_rate', min\_value=0.2, max\_value=0.5, step=0.1)))

model.add(Dense(100, activation='softmax'))

model.compile(optimizer=hp.Choice('optimizer', values=['adam', 'rmsprop', 'sgd']),

loss='categorical\_crossentropy',

metrics=['accuracy'])

return model

tuner = RandomSearch(

build\_model,

objective='val\_accuracy',

max\_trials=10,

executions\_per\_trial=3,

directory='my\_dir',

project\_name='cifar100')

tuner.search(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))

best\_model = tuner.get\_best\_models(num\_models=1)[0]

# Use callback functions for automated training

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=3, min\_lr=0.00001)

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5)

model\_checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_loss', save\_best\_only=True)

callbacks = [reduce\_lr, early\_stopping, model\_checkpoint]

history = best\_model.fit(x\_train, y\_train, epochs=25, batch\_size=64, validation\_data=(x\_test, y\_test), callbacks=callbacks)

# Generate confusion matrix plot

y\_pred = best\_model.predict(x\_test)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_true = np.argmax(y\_test, axis=1)

conf\_matrix = confusion\_matrix(y\_true, y\_pred\_classes)

plt.figure(figsize=(10, 10))

sns.heatmap(conf\_matrix, annot=True, fmt='d')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Plot training and testing loss and accuracy

def plot\_metrics(history):

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# Plot loss

ax1.plot(history.history['loss'], label='train\_loss')

ax1.plot(history.history['val\_loss'], label='val\_loss')

ax1.set\_title('Training and Validation Loss')

ax1.set\_xlabel('Epochs')

ax1.set\_ylabel('Loss')

ax1.legend()

# Plot accuracy

ax2.plot(history.history['accuracy'], label='train\_accuracy')

ax2.plot(history.history['val\_accuracy'], label='val\_accuracy')

ax2.set\_title('Training and Validation Accuracy')

ax2.set\_xlabel('Epochs')

ax2.set\_ylabel('Accuracy')

ax2.legend()

plt.show()

plot\_metrics(history)

# Additional visualizations

# Visualization 1: Accuracy Per Class

accuracy\_per\_class = conf\_matrix.diagonal() / conf\_matrix.sum(axis=1)

plt.figure(figsize=(15, 5))

plt.bar(range(100), accuracy\_per\_class)

plt.xlabel('Class')

plt.ylabel('Accuracy')

plt.title('Accuracy Per Class')

plt.show()

# Visualization 2: Misclassified Images

misclassified\_idx = np.where(y\_pred\_classes != y\_true)[0]

fig, axes = plt.subplots(3, 3, figsize=(10, 10))

axes = axes.ravel()

for i in np.arange(0, 9):

idx = misclassified\_idx[i]

axes[i].imshow(x\_test[idx])

axes[i].set\_title(f'True: {y\_true[idx]}, Pred: {y\_pred\_classes[idx]}')

axes[i].axis('off')

plt.subplots\_adjust(wspace=0.5)

plt.show()

**Output:**

Trial 10 Complete [00h 05m 34s]

val\_accuracy: 0.3841666678587596

Best val\_accuracy So Far: 0.40450000762939453

Total elapsed time: 00h 49m 37s

Epoch 1/25

782/782 [==============================] - 7s 8ms/step - loss: 1.2413 - accuracy: 0.6419 - val\_loss: 2.5126 - val\_accuracy: 0.4220 - lr: 0.0010

Epoch 2/25

21/782 [..............................] - ETA: 3s - loss: 1.0331 - accuracy: 0.6949

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

782/782 [==============================] - 5s 6ms/step - loss: 1.1512 - accuracy: 0.6646 - val\_loss: 2.5804 - val\_accuracy: 0.4137 - lr: 0.0010

Epoch 3/25

782/782 [==============================] - 5s 6ms/step - loss: 1.0674 - accuracy: 0.6852 - val\_loss: 2.6999 - val\_accuracy: 0.4087 - lr: 0.0010

Epoch 4/25

782/782 [==============================] - 5s 6ms/step - loss: 0.9991 - accuracy: 0.7001 - val\_loss: 2.7928 - val\_accuracy: 0.4071 - lr: 0.0010

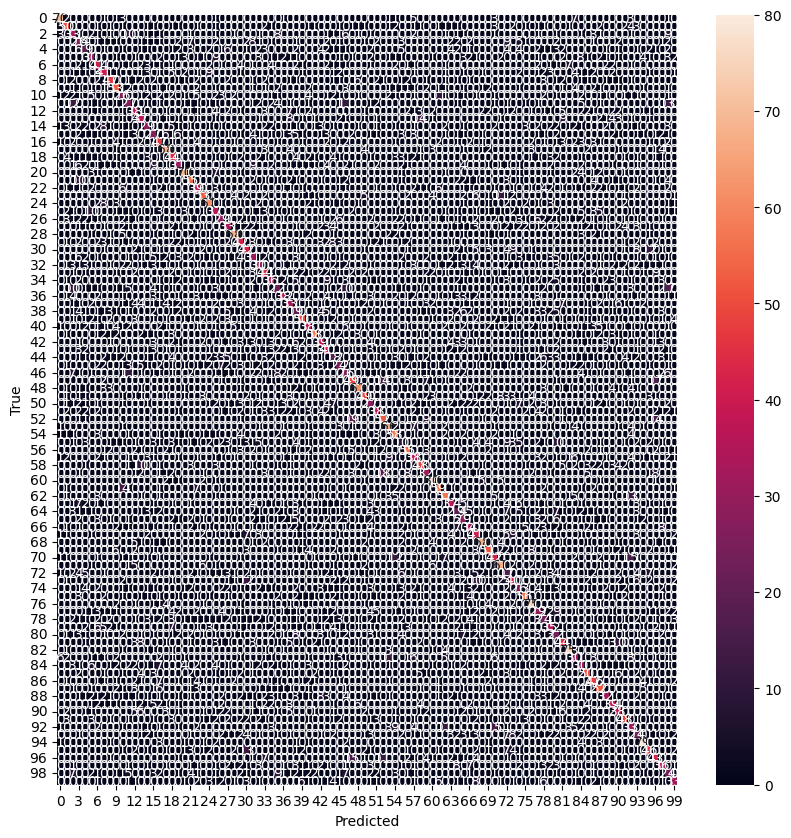
Epoch 5/25

782/782 [==============================] - 4s 6ms/step - loss: 0.7465 - accuracy: 0.7741 - val\_loss: 2.8415 - val\_accuracy: 0.4196 - lr: 2.0000e-04

Epoch 6/25

782/782 [==============================] - 5s 6ms/step - loss: 0.6805 - accuracy: 0.7949 - val\_loss: 2.8785 - val\_accuracy: 0.4155 - lr: 2.0000e-04

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



A comparison of a graph

Description automatically generated with medium confidence

A graph of blue vertical lines

Description automatically generated

A screenshot of a photo

Description automatically generated