Neural Network Models for Object Recognition Presentation

Slide 1 – Introduction

Hello and welcome to my presentation on developing a Neural Network Model for object recognition using the CIFAR-10 image dataset for object recognition using Python. This will be done by taking apart the process of a neural network and making it work for analysing and training a model to analyse images in CIFAR-10 and accurately put them in the right class for example if a truck is a truck. We will breakdown layer by layer how this model is made and what functions it uses in order to do this.

Slide 2 – Cleaning the Data

The CIFAR-10 dataset when downloaded is split into 5 batches to prevent the user's computer running out of memory but for our use we will be combining them so we can use the data in its entirety (Chansung, 2018). However, to combine these we first need to unpack them as they were packed using the pickle module so first, we will depickle using code provided by the Canadian Institute For Advance Research then using the package NumPy to combine them. To partition the dataset we used "train_test_split" from scikit-learn to get the 80/20 split for training and validation, in that order.

We chose "train_test_split" for its simplicity and handles things such as random shuffling and splitting of the data in a single rather than multiple step process. It is also has a "random state" parameter which is easily repeatable which is great for consistency in this process.

With 60,000 images with 10,000 already set aside for testing we will split the 50,000 remaining into a training and a validation set using a split of 80/20, as mentioned before, so the training set will have a size of 40,000 and the validation set will have a 10,000 set size. More on validation later.

Slide 3 – Metadata

A quick rundown into the metadata of this dataset is that there are overall 10 classes which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck with 6,000 images in each class which are all 32 by 32 by 3 images as shown on slide 3 which are not to scale on the slide.

Slide 4 – Importance of maintaining a separate Validation Set

A validation set is imperative when developing a neural network model for object recognition because it ensures that the model remains unbiased when being evaluated. This is done by effectively not letting the model see this data whilst it is being trained. This allows for the hyperparameters to be tuned which are manually set when we will be training this machine learning model. Also noting that hyperparameters are different to parameters that come from the data itself as hyperparameters are set by the user. (Ebner, 2023), (Wilber & Werness, 2022)

During this process we also need to reduce overfitting which occurs when the model we are training matches the dataset we are training's model too closely. If this happens the model can become biased and cannot make accurate conclusions from any other data besides the data, it was trained on. Therefore, when we check this data against the validation set, we will get an unbiased output, we also want to avoid underfitting which can lead to unreliable results so ultimately the validation set allows us to find the goldilocks zone of sorts to find the right fit to put things simply. (Ebner, 2023), (Wilber & Werness, 2022)

Slide 5 – Architecture of the Artificial Neural Network

Artificial Neural Networks are made up of layers input, convolutional, fully connected and output. Which have been used to build this neural network.

Input is used to abide by the CIFAR-10 images which are 32 by 32 pixel images which is where the data enters and checks to make sure they meet those specifications. This is where weights are assigned and give importance to the variables (IBM, 2021). This is structured as a 32 by 32 by 3 neurons as seen in appendix 2.

Convolutional neural networks are used typically for image recognition which is useful for CIFAR-10 as it is a image based dataset where we are trying to find patterns. This is also seen as the "Core building block" of a convolutional neural network as it is where most of the computation occurs. This is because it will look at all of the dimensions of the images in CIFAR-10 for its height width and depth (IBM, 2021). The structure for this is the following: first convolutional layer with 32 filters and a 3x3 kernel size; second convolutional layer with 64 filters and a 3x3 kernel size; third convolutional layer with 128 filters and a 3x3 kernel size with each of these 3 layers having ReLU activation. Following this there is "MaxPooling" which comes after all of these layers in order to prevent overfitting and reduce the spatial dimensions (DeepAI, 2019).

Full connected layers happen after the convolutional neural network which it takes input from and is the penultimate layer before the final output layer. Based on the previous layers it will classify the node (IBM, 2021). This is structured as two dense layers: first dense layer is 128 neurons and the second dense layer is 64 neurons.

The output layer is the final layer of a neural network as it shows the final results of the model. The purpose for this is to create a probability score for the 10 classes listed previously in order to give an idea of which image belongs to each class and it will choose the class with the highest probability of it being for example if it is a truck it will give a prediction that it is a truck based on the model developed. This will be based on if the node is above the threshold value and this will then activate it and this is done efficiently with the use of ReLU in earlier layers (IBM, 2021). The output layer is structured into two a dense layer of 10 neurons with softmax activation which allows the output to be turned into a probability distribution for the 10 classes, more on softmax activation later (DeepAI, 2019).

Slide 6 – Chosen Activation Function

"Activation functions are what decide if a neuron should be activated or not based on the weighted sum and further adding bias to it" (Tiwari, 2024). You then need to choose an activation function and for this use case we chose Rectified Linear Units known as ReLU. This is because it is simple and efficient compared to others such as sigmoid. It also only activates a subset of neurons at any given time adding to the efficiency mentioned earlier which helps it generalise (McQuillan, 2022). This function is used in Appendix 2 in the connected, outer and hidden layers of the neural network to help with the vanishing gradient problem which is when more layers use certain activation functions the gradient loss function approaches zero which makes the model harder to train which in our case would be hard as we are trying to train a model (Wang, 2019). We also use a softmax function on the outer layer which does a probability distribution over our 10 classes which is good for these kind of tasks (Priya, 2023).

Slide 7 – Loss Function Implementation

The loss function implanted was “Categorical Cross-Entropy Loss” which is common for when you have a problem with multiple classes where it compares the class label with the predicted probability which helps to make the model more efficient when training (365 Data Science, 2023). This is helpful for us as CIFAR-10 uses 10 classes and this is the use case for Categorical Cross-Entropy Loss. We also use this as it is compatible with both Adam and softmax activation as it measures how well softmax did with labelling so if a truck was actually a truck.

Slide 8 – Number of epochs utilised in the modelling process

Now we will discuss the number of epochs. Epochs are effectively a training iteration where it goes through the entire training process and in this case, we did 10 and built up to 50 epochs. This is because each epoch learns from the last and gradually improves its performance over the 50 training cycles as it changes its weighting of different losses. It is also a good idea when a dataset is so big that you run multiple epochs so that it can workout the patterns. (Brownlee, 2022)

As mentioned in the validation set section we want to prevent over and under fitting. All 50 epochs were ran but a “early stopping” line of code was implemented into the model so that it would stop early if it noticed any over/under fitting if there was not any validation loss improvement during the 50 epochs. 50 is also at a high enough number of training cycles to do enough testing and be able to run on an average users computer at 50 epochs takes reasonably high computing power depending on your CPU or GPU. (Kishore, 2024)

Slide 9 – Neural Network's Design Elements Strategy

The design elements used for this neural network were to use regularisation in the form of dropout layers, batch normalisation and an optimiser in the form of Adaptive Moment Estimation also known as Adam.

The regularisation dropout layers is used to help prevent overfitting in a neural network by randomly setting a fraction of inputs which helps with training the model to not get used to various specific neurons allowing for generalisation (Keras Documentation, No Date). These are added after the convolutional layers.

Batch normalisation is used to make neural networks faster and increase stability by increasing the layers to the neural network which standardises and normalises the previous layer (Saxena, 2024). This part is nestled in between the convolutional layer and the activation function.

The optimiser used is adaptive moment estimation as mentioned also called Adam where it uses stochastic gradient descent which helps optimise random variables (Vishwakarma, 2024). In the structure this is used in the model compilation stage.

Slide 10 – Reflection

I learned a lot during this process and getting to use CIFAR-10 was intimidating at first and training a model took a lot to wrap my head around but overall, a satisfying process. Understanding how much went into a neural network with the multiple layers from start to finish was interesting to experiment with as there is so much to each layer with there being multiple filters and denseness of layers. Learning how all of these factors comes together to help the model learn and to be general and avoid being biased and be a functional model. Getting to grips with so many tiny details such as the vanishing gradient problem and deciding with activation function was best to combat this such as ReLU. Overall, a neural network is very complex and it was an interesting process to take apart these layers and read deeply into them and understand each stage.

<u>Reference List</u>

365 Data Science (2023) *What is cross-entropy loss function?*, *365 Data Science*. Available at: https://365datascience.com/tutorials/machine-learning-tutorials/cross-entropy-loss/ (Accessed: 19 May 2024).

Brownlee, J. (2022) *Difference between a batch and an epoch in a neural network*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ (Accessed: 20 May 2024).

Chansung, P. (2018) *CIFAR-10 image classification in tensorflow*, *Medium*. Available at: https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c (Accessed: 15 May 2024).

DeepAI (2019) *Max pooling*, *DeepAI*. Available at: https://deepai.org/machine-learning-glossary-and-terms/max-pooling#:~:text=Max%20pooling%20is%20performed%20on,maximum%20value%20within%20the%20window. (Accessed: 20 May 2024).

Ebner, J. (2023) *Training, validation, and test sets … explained*, *Sharp Sight*. Available at: https://www.sharpsightlabs.com/blog/training-validation-and-test-sets/ (Accessed: 16 May 2024).

IBM (2021) *What is a neural network?*, *IBM*. Available at: https://www.ibm.com/topics/neural-networks (Accessed: 15 May 2024).

Keras Documentation (no date) *Keras documentation: Dropout layer*, *Keras*. Available at: https://keras.io/api/layers/regularization_layers/dropout/ (Accessed: 20 May 2024).

Kishore (2024) *Understanding epochs in neural networks: A comprehensive guide*, *Explore Learn Grow*. Available at: https://blog.cogxta.com/understanding-epochs-in-neural-networks-a-comprehensive-guide/ (Accessed: 20 May 2024).

Krizhevsky, A. (2009) *The CIFAR-10 dataset*, *CIFAR-10 and CIFAR-100 datasets*. Available at: https://www.cs.toronto.edu/~kriz/cifar.html (Accessed: 10 May 2024).

McQuillan, L. (2022) *Deep learning 101: Transformer activation functions explainer - sigmoid, Relu, Gelu, swish*, *Salt Data Labs*. Available at: https://www.saltdatalabs.com/blog/deep-learning-101-transformer-activation-functions-explainer-relu-leaky-relu-gelu-elu-selu-softmax-and-more (Accessed: 20 May 2024).

Priya, B. (2023) *Softmax activation function: Everything you need to know*, *Pinecone*. Available at: https://www.pinecone.io/learn/softmax-activation/ (Accessed: 27 May 2024).

Saxena, S. (2024) *Introduction to batch normalization*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/#h-what-is-batch-normalization (Accessed: 20 May 2024).

Tiwari, S. (2024) *Activation functions in neural networks*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/activation-functions-neural-networks/#:~:text=The%20activation%20function%20decides%20whether,the%20output%20of%20a%20neuron. (Accessed: 17 May 2024).

Vishwakarma, N. (2024) *What is Adam Optimizer?*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/ (Accessed: 27 May 2024).

Wang, C.-F. (2019) *The vanishing gradient problem*, *Medium*. Available at: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 (Accessed: 20 May 2024).

Wilber, J. and Werness, B. (2022) *Train, test, and validation sets*, *MLU*. Available at: https://mlu-explain.github.io/train-test-validation/ (Accessed: 15 May 2024).

Appendices

Appendix 1

Input:

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Step 2: Import necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split
import os
import pickle
from tensorflow.keras.utils import to_categorical

# Step 3: Define the path to the directory containing your data files
data_dir = '/content/drive/MyDrive/Machine Learning'

# Step 4: Function to unpickle the data
def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

# Step 5: Load the data batches
batch1 = os.path.join(data_dir, 'data_batch_1')
data_batch_1 = unpickle(batch1)
```

```python
batch2 = os.path.join(data_dir, 'data_batch_2')
data_batch_2 = unpickle(batch2)

batch3 = os.path.join(data_dir, 'data_batch_3')
data_batch_3 = unpickle(batch3)

batch4 = os.path.join(data_dir, 'data_batch_4')
data_batch_4 = unpickle(batch4)

batch5 = os.path.join(data_dir, 'data_batch_5')
data_batch_5 = unpickle(batch5)

# Step 6: Combine the data and labels from all batches
X_train = np.vstack((data_batch_1[b'data'], data_batch_2[b'data'],
data_batch_3[b'data'], data_batch_4[b'data'], data_batch_5[b'data']))
y_train = np.hstack((data_batch_1[b'labels'], data_batch_2[b'labels'],
data_batch_3[b'labels'], data_batch_4[b'labels'],
data_batch_5[b'labels']))

# Step 7: Reshape data to 32x32x3
X_train = X_train.reshape(-1, 32, 32, 3)

# Step 8: Normalize pixel values to be between 0 and 1
X_train = X_train.astype('float32') / 255.0

# Step 9: Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

# Step 10: One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_val = to_categorical(y_val, 10)

# Step 11: Print shapes of the resulting arrays
print(f"Training data shape: {X_train.shape}")
print(f"Validation data shape: {X_val.shape}")
print(f"Training labels shape: {y_train.shape}")
print(f"Validation labels shape: {y_val.shape}")
```
Output:

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
Training data shape: (40000, 32, 32, 3)
Validation data shape: (10000, 32, 32, 3)
Training labels shape: (40000, 10)
Validation labels shape: (10000, 10)
```

Appendix 2

Input:

```
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Assuming X_train, X_val, y_train, and y_val are already defined and
available

# Define the model
model = Sequential()

# First Convolutional Layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

# Second Convolutional Layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

# Third Convolutional Layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten the output
model.add(Flatten())

# Fully Connected Layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))

# Output Layer
model.add(Dense(10, activation='softmax'))

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

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
```

```
# Define data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

# Fit the data generator on the training data
datagen.fit(X_train)

# Print the model summary
model.summary()

# Train the model with data augmentation and early stopping
history = model.fit(datagen.flow(X_train, y_train, batch_size=64),
                    epochs=50,
                    validation_data=(X_val, y_val),
                    verbose=2,
                    steps_per_epoch=len(X_train) // 64,
                    callbacks=[early_stopping])

# Evaluate the model on the validation set
val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=2)
print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy}')
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 15, 15, 32)        0         
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 6, 6, 64)          0         
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 128)         73856     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 128)         0         
 g2D)                                                            
 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 128)               65664

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 167818 (655.54 KB)
Trainable params: 167818 (655.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/50
625/625 - 26s - loss: 2.1020 - accuracy: 0.2045 - val_loss: 1.9615 -
val_accuracy: 0.2715 - 26s/epoch - 42ms/step
Epoch 2/50
625/625 - 20s - loss: 1.8977 - accuracy: 0.2906 - val_loss: 1.8854 -
val_accuracy: 0.2928 - 20s/epoch - 32ms/step
Epoch 3/50
625/625 - 20s - loss: 1.8401 - accuracy: 0.3158 - val_loss: 1.8219 -
val_accuracy: 0.3304 - 20s/epoch - 33ms/step
Epoch 4/50
625/625 - 20s - loss: 1.7953 - accuracy: 0.3373 - val_loss: 1.7224 -
val_accuracy: 0.3648 - 20s/epoch - 32ms/step
Epoch 5/50
625/625 - 20s - loss: 1.7628 - accuracy: 0.3520 - val_loss: 1.7026 -
val_accuracy: 0.3827 - 20s/epoch - 32ms/step
Epoch 6/50
625/625 - 20s - loss: 1.7339 - accuracy: 0.3679 - val_loss: 1.7535 -
val_accuracy: 0.3680 - 20s/epoch - 32ms/step
Epoch 7/50
625/625 - 20s - loss: 1.7155 - accuracy: 0.3767 - val_loss: 1.6801 -
val_accuracy: 0.3824 - 20s/epoch - 32ms/step
Epoch 8/50
625/625 - 20s - loss: 1.6947 - accuracy: 0.3828 - val_loss: 1.6007 -
val_accuracy: 0.4245 - 20s/epoch - 32ms/step
Epoch 9/50
625/625 - 20s - loss: 1.6726 - accuracy: 0.3944 - val_loss: 1.5472 -
val_accuracy: 0.4380 - 20s/epoch - 32ms/step
Epoch 10/50
625/625 - 20s - loss: 1.6560 - accuracy: 0.3999 - val_loss: 1.6427 -
val_accuracy: 0.4065 - 20s/epoch - 32ms/step
Epoch 11/50
625/625 - 20s - loss: 1.6401 - accuracy: 0.4052 - val_loss: 1.5960 -
val_accuracy: 0.4275 - 20s/epoch - 32ms/step
Epoch 12/50
625/625 - 20s - loss: 1.6272 - accuracy: 0.4123 - val_loss: 1.6405 -
val_accuracy: 0.4077 - 20s/epoch - 32ms/step
Epoch 13/50
625/625 - 20s - loss: 1.6189 - accuracy: 0.4171 - val_loss: 1.6759 -
val_accuracy: 0.4102 - 20s/epoch - 32ms/step
Epoch 14/50
625/625 - 20s - loss: 1.6140 - accuracy: 0.4172 - val_loss: 1.5558 -
val_accuracy: 0.4380 - 20s/epoch - 33ms/step
```

```
Epoch 15/50
625/625 - 20s - loss: 1.5986 - accuracy: 0.4248 - val_loss: 1.5760 -
val_accuracy: 0.4371 - 20s/epoch - 32ms/step
Epoch 16/50
625/625 - 20s - loss: 1.5918 - accuracy: 0.4257 - val_loss: 1.5818 -
val_accuracy: 0.4356 - 20s/epoch - 32ms/step
Epoch 17/50
625/625 - 20s - loss: 1.5885 - accuracy: 0.4268 - val_loss: 1.4969 -
val_accuracy: 0.4665 - 20s/epoch - 32ms/step
Epoch 18/50
625/625 - 20s - loss: 1.5774 - accuracy: 0.4308 - val_loss: 1.5709 -
val_accuracy: 0.4238 - 20s/epoch - 32ms/step
Epoch 19/50
625/625 - 20s - loss: 1.5694 - accuracy: 0.4342 - val_loss: 1.5405 -
val_accuracy: 0.4379 - 20s/epoch - 32ms/step
Epoch 20/50
625/625 - 20s - loss: 1.5618 - accuracy: 0.4430 - val_loss: 1.5580 -
val_accuracy: 0.4440 - 20s/epoch - 32ms/step
Epoch 21/50
625/625 - 20s - loss: 1.5547 - accuracy: 0.4439 - val_loss: 1.5275 -
val_accuracy: 0.4560 - 20s/epoch - 32ms/step
Epoch 22/50
625/625 - 20s - loss: 1.5498 - accuracy: 0.4435 - val_loss: 1.4618 -
val_accuracy: 0.4675 - 20s/epoch - 32ms/step
Epoch 23/50
625/625 - 20s - loss: 1.5473 - accuracy: 0.4483 - val_loss: 1.4559 -
val_accuracy: 0.4732 - 20s/epoch - 32ms/step
Epoch 24/50
625/625 - 20s - loss: 1.5418 - accuracy: 0.4511 - val_loss: 1.4925 -
val_accuracy: 0.4595 - 20s/epoch - 32ms/step
Epoch 25/50
625/625 - 20s - loss: 1.5383 - accuracy: 0.4498 - val_loss: 1.4297 -
val_accuracy: 0.4820 - 20s/epoch - 32ms/step
Epoch 26/50
625/625 - 20s - loss: 1.5406 - accuracy: 0.4458 - val_loss: 1.4542 -
val_accuracy: 0.4693 - 20s/epoch - 32ms/step
Epoch 27/50
625/625 - 20s - loss: 1.5238 - accuracy: 0.4555 - val_loss: 1.4843 -
val_accuracy: 0.4532 - 20s/epoch - 32ms/step
Epoch 28/50
625/625 - 20s - loss: 1.5210 - accuracy: 0.4580 - val_loss: 1.4353 -
val_accuracy: 0.4847 - 20s/epoch - 32ms/step
Epoch 29/50
625/625 - 20s - loss: 1.5245 - accuracy: 0.4533 - val_loss: 1.4284 -
val_accuracy: 0.4860 - 20s/epoch - 32ms/step
Epoch 30/50
625/625 - 20s - loss: 1.5171 - accuracy: 0.4558 - val_loss: 1.4306 -
val_accuracy: 0.4858 - 20s/epoch - 32ms/step
Epoch 31/50
625/625 - 20s - loss: 1.5060 - accuracy: 0.4595 - val_loss: 1.4050 -
val_accuracy: 0.4970 - 20s/epoch - 32ms/step
Epoch 32/50
625/625 - 20s - loss: 1.5126 - accuracy: 0.4594 - val_loss: 1.4500 -
val_accuracy: 0.4782 - 20s/epoch - 32ms/step
Epoch 33/50
625/625 - 20s - loss: 1.5070 - accuracy: 0.4619 - val_loss: 1.4937 -
val_accuracy: 0.4609 - 20s/epoch - 32ms/step
Epoch 34/50
```

```
625/625 - 20s - loss: 1.5039 - accuracy: 0.4601 - val_loss: 1.4923 -
val_accuracy: 0.4591 - 20s/epoch - 32ms/step
Epoch 35/50
625/625 - 20s - loss: 1.5012 - accuracy: 0.4645 - val_loss: 1.4571 -
val_accuracy: 0.4707 - 20s/epoch - 32ms/step
Epoch 36/50
625/625 - 20s - loss: 1.4953 - accuracy: 0.4624 - val_loss: 1.4057 -
val_accuracy: 0.4912 - 20s/epoch - 32ms/step
Epoch 37/50
625/625 - 20s - loss: 1.4923 - accuracy: 0.4706 - val_loss: 1.4516 -
val_accuracy: 0.4813 - 20s/epoch - 33ms/step
Epoch 38/50
625/625 - 20s - loss: 1.4917 - accuracy: 0.4686 - val_loss: 1.3836 -
val_accuracy: 0.5041 - 20s/epoch - 32ms/step
Epoch 39/50
625/625 - 20s - loss: 1.4869 - accuracy: 0.4733 - val_loss: 1.4503 -
val_accuracy: 0.4755 - 20s/epoch - 32ms/step
Epoch 40/50
625/625 - 20s - loss: 1.4883 - accuracy: 0.4722 - val_loss: 1.4147 -
val_accuracy: 0.4865 - 20s/epoch - 32ms/step
Epoch 41/50
625/625 - 20s - loss: 1.4839 - accuracy: 0.4726 - val_loss: 1.4260 -
val_accuracy: 0.4891 - 20s/epoch - 32ms/step
Epoch 42/50
625/625 - 20s - loss: 1.4770 - accuracy: 0.4762 - val_loss: 1.4818 -
val_accuracy: 0.4621 - 20s/epoch - 32ms/step
Epoch 43/50
625/625 - 20s - loss: 1.4784 - accuracy: 0.4772 - val_loss: 1.4937 -
val_accuracy: 0.4670 - 20s/epoch - 32ms/step
Epoch 44/50
625/625 - 20s - loss: 1.4724 - accuracy: 0.4738 - val_loss: 1.4813 -
val_accuracy: 0.4614 - 20s/epoch - 32ms/step
Epoch 45/50
625/625 - 20s - loss: 1.4686 - accuracy: 0.4763 - val_loss: 1.3873 -
val_accuracy: 0.5008 - 20s/epoch - 32ms/step
Epoch 46/50
625/625 - 20s - loss: 1.4697 - accuracy: 0.4748 - val_loss: 1.4012 -
val_accuracy: 0.4929 - 20s/epoch - 32ms/step
Epoch 47/50
625/625 - 20s - loss: 1.4720 - accuracy: 0.4761 - val_loss: 1.3608 -
val_accuracy: 0.5094 - 20s/epoch - 32ms/step
Epoch 48/50
625/625 - 21s - loss: 1.4706 - accuracy: 0.4758 - val_loss: 1.4215 -
val_accuracy: 0.4830 - 21s/epoch - 33ms/step
Epoch 49/50
625/625 - 20s - loss: 1.4645 - accuracy: 0.4812 - val_loss: 1.3844 -
val_accuracy: 0.5028 - 20s/epoch - 32ms/step
Epoch 50/50
625/625 - 20s - loss: 1.4647 - accuracy: 0.4789 - val_loss: 1.4129 -
val_accuracy: 0.4915 - 20s/epoch - 32ms/step
313/313 - 1s - loss: 1.4129 - accuracy: 0.4915 - 640ms/epoch - 2ms/step
Validation Loss: 1.4128721952438354
Validation Accuracy: 0.49149999022483826
```