Loading Libraries

In [2]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

```
WARNING:tensorflow:From c:\Users\rodri\AppData\Local\Programs\Python\Python39\lib\sit
e-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy
is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Loading Data

In [3]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data(

# Normalize pixel values to be between 0 and 1 by dividing by 255
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Training the Models

Model 1: Artificial Neural Networks (ANNs)

In this model exploration, I aim to optimize the performance of an image classification model using Artificial Neural Networks (ANNs) by adjusting various configurations and hyperparameters. The initial focus will be on exploring and comparing different optimizer algorithms, which play a critical role in determining how the network learns from the data by adjusting its weights during the training process.

Optimizers

Optimizers in ANNs are pivotal in refining the model's parameters concerning a defined loss function, essentially guiding the learning process by updating the weights iteratively. For this investigation, I'll delve into three distinct optimizer algorithms:

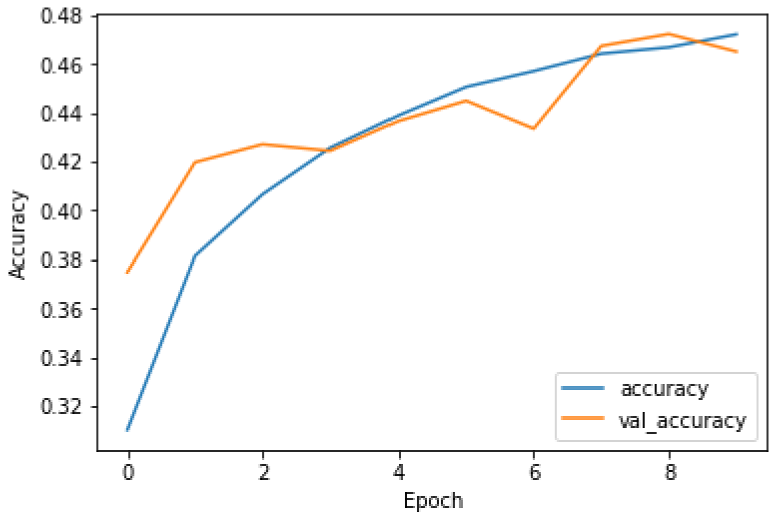
- Adam (Adaptive Moment Estimation): This algorithm combines the advantages of both Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) approaches. It maintains adaptive learning rates for each parameter, adjusting them as the training progresses based on the moments of gradients.
- SGD (Stochastic Gradient Descent): This classic optimization algorithm updates the model's parameters proportional to the gradient of the error with respect to a single training example, often utilizing a learning rate to determine the step size in the parameter space.

- RMSProp (Root Mean Square Propagation): RMSProp adjusts the learning rates for different model parameters by dividing the learning rate for a weight by the average of the magnitudes of recent gradients for that weight.

In [4]:

```
# Create ANN model with the Adam optimizer
ann = models.Sequential([
    layers.Flatten(input_shape = (32,32,3)),
    layers.Dense(128, activation = 'relu'),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(10, activation = 'softmax')
])

ann.compile(optimizer = 'adam',
            loss = 'sparse_categorical_crossentropy',
           metrics = ['accuracy'])

# Train the model
history = ann.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = ann.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

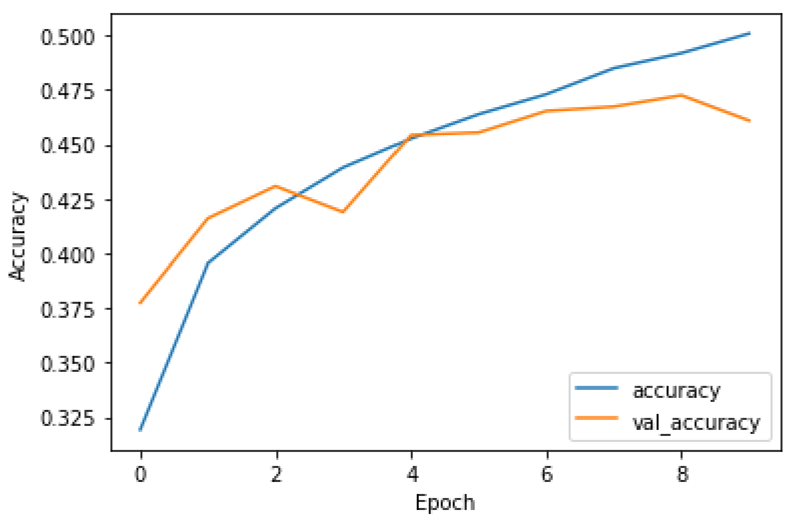
```
WARNING:tensorflow:From c:\Users\rodri\AppData\Local\Programs\Python\Python39\lib\sit
e-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Ple
ase use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\rodri\AppData\Local\Programs\Python\Python39\lib\sit
e-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is depre
cated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/10
WARNING:tensorflow:From c:\Users\rodri\AppData\Local\Programs\Python\Python39\lib\sit
e-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is d
eprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\Users\rodri\AppData\Local\Programs\Python\Python39\lib\sit
e-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_ou
tside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_func
tions instead.

1563/1563 [==============================] - 6s 3ms/step - loss: 1.8998 - accuracy:
0.3103 - val_loss: 1.7608 - val_accuracy: 0.3748
Epoch 2/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7240 - accuracy:
0.3814 - val_loss: 1.6405 - val_accuracy: 0.4197
Epoch 3/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.6530 - accuracy:
0.4066 - val_loss: 1.6171 - val_accuracy: 0.4271
Epoch 4/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.6082 - accuracy:
0.4256 - val_loss: 1.6199 - val_accuracy: 0.4244
Epoch 5/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5701 - accuracy:
0.4389 - val_loss: 1.5792 - val_accuracy: 0.4366
Epoch 6/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5429 - accuracy:
0.4506 - val_loss: 1.5529 - val_accuracy: 0.4449
Epoch 7/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5229 - accuracy:
0.4571 - val_loss: 1.5820 - val_accuracy: 0.4335
Epoch 8/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5026 - accuracy:
0.4642 - val_loss: 1.4944 - val_accuracy: 0.4674
Epoch 9/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4877 - accuracy:
0.4668 - val_loss: 1.4859 - val_accuracy: 0.4723
Epoch 10/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4752 - accuracy:
0.4722 - val_loss: 1.5135 - val_accuracy: 0.4651
313/313 [==============================] - 0s 1ms/step - loss: 1.5135 - accuracy: 0.4
651
Test accuracy: 0.4650999903678894
```



In [4]:

```
# Create ANN model with the SGD optimizer
ann = models.Sequential([
    layers.Flatten(input_shape = (32,32,3)),
    layers.Dense(128, activation = 'relu'),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(10, activation = 'softmax')
])

ann.compile(optimizer = 'sgd',
            loss = 'sparse_categorical_crossentropy',
           metrics = ['accuracy'])

# Train the model
history = ann.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = ann.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/10
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8969 - accuracy:
0.3192 - val_loss: 1.7575 - val_accuracy: 0.3774
Epoch 2/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.7102 - accuracy:
0.3954 - val_loss: 1.6654 - val_accuracy: 0.4160
Epoch 3/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.6349 - accuracy:
0.4206 - val_loss: 1.6157 - val_accuracy: 0.4308
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5811 - accuracy:
0.4393 - val_loss: 1.6265 - val_accuracy: 0.4189
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5424 - accuracy:
0.4526 - val_loss: 1.5442 - val_accuracy: 0.4542
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5121 - accuracy:
0.4638 - val_loss: 1.5240 - val_accuracy: 0.4554
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4834 - accuracy:
0.4730 - val_loss: 1.5039 - val_accuracy: 0.4653
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4561 - accuracy:
0.4849 - val_loss: 1.4865 - val_accuracy: 0.4673
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4341 - accuracy:
0.4919 - val_loss: 1.4912 - val_accuracy: 0.4725
Epoch 10/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4139 - accuracy:
0.5008 - val_loss: 1.5458 - val_accuracy: 0.4609
313/313 [==============================] - 0s 949us/step - loss: 1.5458 - accuracy:
0.4609
Test accuracy: 0.4609000086784363
```

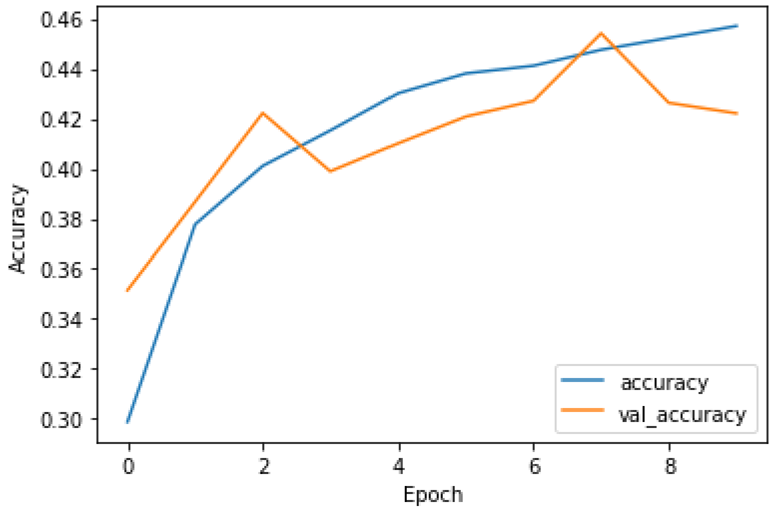


In [5]:

```
# Create ANN model with the RMSProp optimizer
ann = models.Sequential([
    layers.Flatten(input_shape = (32,32,3)),
    layers.Dense(128, activation = 'relu'),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(10, activation = 'softmax')
])

ann.compile(optimizer = 'rmsprop',
            loss = 'sparse_categorical_crossentropy',
```

```python
          metrics = ['accuracy'])

# Train the model
history = ann.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = ann.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.9302 - accuracy:
0.2988 - val_loss: 1.7885 - val_accuracy: 0.3515
Epoch 2/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7426 - accuracy:
0.3777 - val_loss: 1.7004 - val_accuracy: 0.3867
Epoch 3/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6792 - accuracy:
0.4010 - val_loss: 1.6222 - val_accuracy: 0.4222
Epoch 4/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6374 - accuracy:
0.4152 - val_loss: 1.6799 - val_accuracy: 0.3989
Epoch 5/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6075 - accuracy:
0.4301 - val_loss: 1.6709 - val_accuracy: 0.4101
Epoch 6/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.5862 - accuracy:
0.4381 - val_loss: 1.6372 - val_accuracy: 0.4208
Epoch 7/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.5672 - accuracy:
0.4411 - val_loss: 1.6096 - val_accuracy: 0.4271
Epoch 8/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.5550 - accuracy:
0.4475 - val_loss: 1.5519 - val_accuracy: 0.4541
Epoch 9/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.5399 - accuracy:
0.4523 - val_loss: 1.6618 - val_accuracy: 0.4263
Epoch 10/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.5319 - accuracy:
0.4570 - val_loss: 1.6914 - val_accuracy: 0.4221
313/313 [==============================] - 0s 995us/step - loss: 1.6914 - accuracy:
0.4221
Test accuracy: 0.4221000075340271
```



Results

A 46.50% accuracy score with the Adam optimizer showcases its performance as the most effective among the three optimizers—Adam, SGD, and RMSProp—utilized in the image classification task. This accuracy metric reflects the model's ability to correctly classify nearly half of the images within the dataset. Adam's success in achieving the highest accuracy suggests its suitability for this specific dataset and neural network configuration. Its adaptive learning rate and combination of momentum techniques seem to have facilitated better convergence and parameter optimization compared to the other optimizers.

Layers

Now, the exploration shifts towards varying the number of layers within the neural network architecture. The number of layers profoundly influences the network's capacity to learn intricate patterns and features within the dataset.

This phase involves experimenting with different layer configurations, such as:

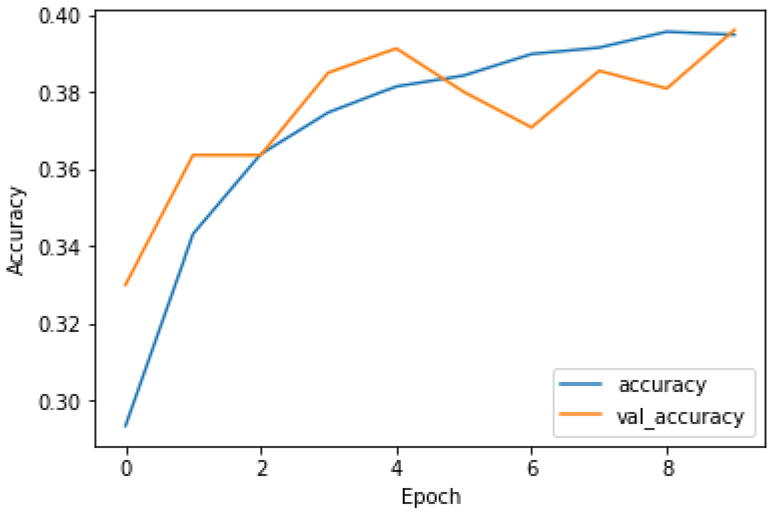
- Shallow Networks: Consisting of fewer layers, typically with a small number of hidden layers.
- Deep Networks: Incorporating a larger number of layers, allowing for a more intricate hierarchy of features and representations.

By systematically adjusting the number of layers while maintaining other hyperparameters constant or within defined ranges, the goal is to observe how the model's accuracy responds to these structural modifications. Each configuration will undergo training and evaluation, measuring accuracy metrics and potentially other performance indicators.

In [6]:
```
# Create a Shallow Networks model
ann = models.Sequential([
    layers.Flatten(input_shape = (32,32,3)),
    layers.Dense(128, activation = 'relu'),
    layers.Dense(10, activation = 'softmax')
])
```

```python
ann.compile(optimizer = 'adam',
            loss = 'sparse_categorical_crossentropy',
           metrics = ['accuracy'])

# Train the model
history = ann.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = ann.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

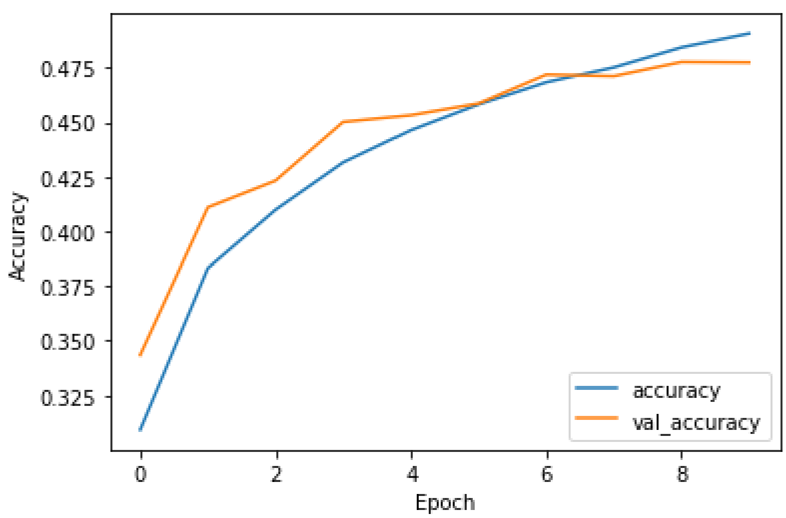
```
Epoch 1/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.9382 - accuracy:
0.2934 - val_loss: 1.8268 - val_accuracy: 0.3300
Epoch 2/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8108 - accuracy:
0.3431 - val_loss: 1.7695 - val_accuracy: 0.3636
Epoch 3/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7589 - accuracy:
0.3638 - val_loss: 1.7479 - val_accuracy: 0.3636
Epoch 4/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7315 - accuracy:
0.3747 - val_loss: 1.7122 - val_accuracy: 0.3850
Epoch 5/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7154 - accuracy:
0.3814 - val_loss: 1.7029 - val_accuracy: 0.3913
Epoch 6/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7053 - accuracy:
0.3843 - val_loss: 1.7167 - val_accuracy: 0.3800
Epoch 7/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6941 - accuracy:
0.3899 - val_loss: 1.7449 - val_accuracy: 0.3708
Epoch 8/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6869 - accuracy:
0.3915 - val_loss: 1.6915 - val_accuracy: 0.3855
Epoch 9/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6803 - accuracy:
0.3957 - val_loss: 1.6943 - val_accuracy: 0.3809
Epoch 10/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6765 - accuracy:
0.3949 - val_loss: 1.6778 - val_accuracy: 0.3961
313/313 [==============================] - 0s 917us/step - loss: 1.6778 - accuracy:
0.3961
Test accuracy: 0.3961000144481659
```



In [7]:

```
# Create a Deep Networks model
ann = models.Sequential([
    layers.Flatten(input_shape = (32,32,3)),
    layers.Dense(256, activation = 'relu'),
    layers.Dense(128, activation = 'relu'),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(32, activation = 'relu'),
    layers.Dense(10, activation = 'softmax')
])

ann.compile(optimizer = 'adam',
            loss = 'sparse_categorical_crossentropy',
           metrics = ['accuracy'])

# Train the model
history = ann.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = ann.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/10
1563/1563 [==============================] - 9s 5ms/step - loss: 1.8968 - accuracy:
0.3092 - val_loss: 1.8106 - val_accuracy: 0.3435
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.7199 - accuracy:
0.3832 - val_loss: 1.6582 - val_accuracy: 0.4110
Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.6422 - accuracy:
0.4098 - val_loss: 1.6291 - val_accuracy: 0.4232
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5831 - accuracy:
0.4316 - val_loss: 1.5516 - val_accuracy: 0.4501
Epoch 5/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5467 - accuracy:
0.4464 - val_loss: 1.5464 - val_accuracy: 0.4532
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5091 - accuracy:
0.4582 - val_loss: 1.5096 - val_accuracy: 0.4586
Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4828 - accuracy:
0.4683 - val_loss: 1.4898 - val_accuracy: 0.4718
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4608 - accuracy:
0.4751 - val_loss: 1.4794 - val_accuracy: 0.4711
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4388 - accuracy:
0.4843 - val_loss: 1.4736 - val_accuracy: 0.4776
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4218 - accuracy:
0.4906 - val_loss: 1.4666 - val_accuracy: 0.4774
313/313 [==============================] - 0s 1ms/step - loss: 1.4666 - accuracy: 0.4
774
Test accuracy: 0.4774000048637390
```



Results

In the context of this comparison, the deeper neural network model demonstrated a higher accuracy score when measured against the shallow network model. This outcome aligns with the typical trend observed in neural network architectures, emphasizing the advantages of leveraging deeper architectures for enhanced learning and better performance in learning complex patterns from the data.

Model 2: Convolutional Neural Networks (CNNs)

Having explored Artificial Neural Networks (ANNs), our attention now shifts to Convolutional Neural Networks (CNNs). These specialized neural networks are tailored for processing structured grid-like data, particularly prevalent in image and video recognition tasks. CNNs utilize convolutional layers, filters, and pooling layers to extract and learn hierarchical representations of features within images. By doing so, they effectively capture spatial hierarchies and patterns present in visual data.

CNNs excel in scenarios where understanding local patterns and spatial relationships is crucial. Their effectiveness in image recognition, object detection, and various computer vision tasks stems from their ability to automatically learn features from raw pixel data. Moreover, CNNs leverage parameter sharing across the input, significantly reducing the number of parameters. This parameter sharing enables efficient learning from high-dimensional inputs like images, contributing to both performance and computational efficiency.

Similar to the experimentation with ANNs, we will explore the impact of different optimizers within CNN architectures to discern their performance differences. Optimizers such as Adam, SGD, RMSProp, and others will be tested to observe their influence on the CNN's learning dynamics, convergence, and overall performance in image-related tasks.

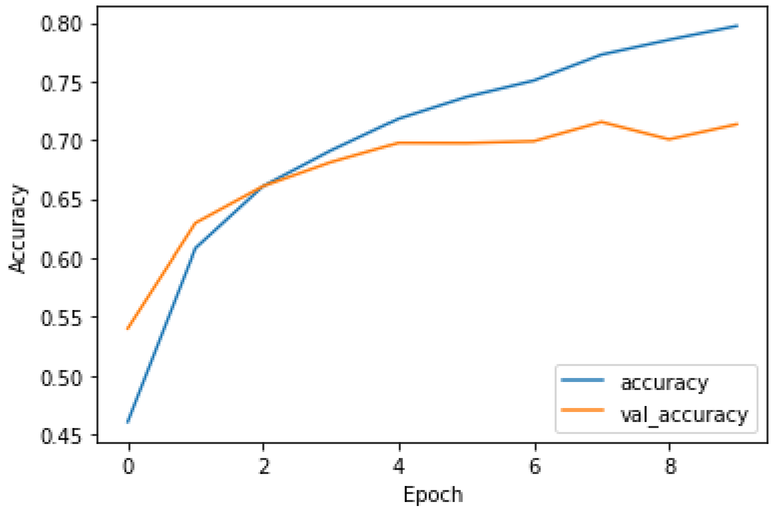
Optimizers

In [8]:

```python
# Create CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax'),  # 10 output classes for CIFAR-10
])

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

```python
# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

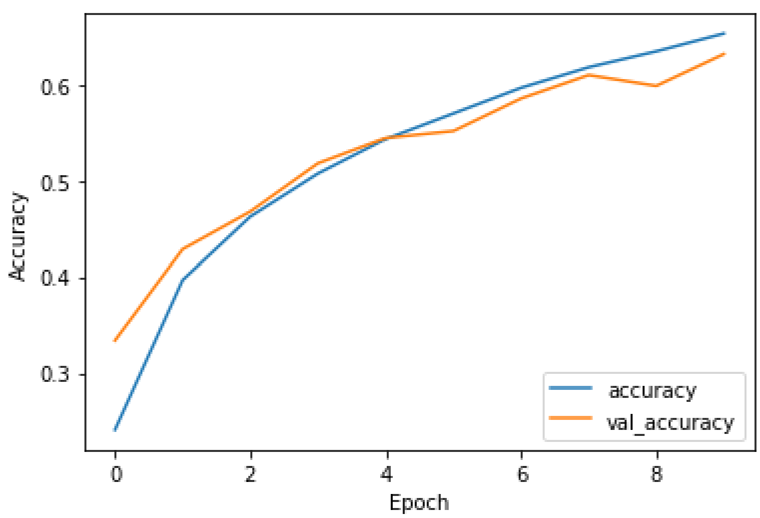
```
WARNING:tensorflow:From c:\Users\rodri\AppData\Local\Programs\Python\Python39\lib\sit
e-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.

Epoch 1/10
1563/1563 [==============================] - 11s 7ms/step - loss: 1.4850 - accuracy:
0.4606 - val_loss: 1.2637 - val_accuracy: 0.5401
Epoch 2/10
1563/1563 [==============================] - 10s 7ms/step - loss: 1.1126 - accuracy:
0.6079 - val_loss: 1.0450 - val_accuracy: 0.6294
Epoch 3/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.9704 - accuracy:
0.6606 - val_loss: 0.9570 - val_accuracy: 0.6607
Epoch 4/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.8827 - accuracy:
0.6910 - val_loss: 0.9103 - val_accuracy: 0.6811
Epoch 5/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.8063 - accuracy:
0.7181 - val_loss: 0.8600 - val_accuracy: 0.6976
Epoch 6/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.7505 - accuracy:
0.7366 - val_loss: 0.8690 - val_accuracy: 0.6974
Epoch 7/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.7014 - accuracy:
0.7506 - val_loss: 0.8765 - val_accuracy: 0.6990
Epoch 8/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.6549 - accuracy:
0.7726 - val_loss: 0.8426 - val_accuracy: 0.7154
Epoch 9/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.6090 - accuracy:
0.7853 - val_loss: 0.8999 - val_accuracy: 0.7006
Epoch 10/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.5727 - accuracy:
0.7970 - val_loss: 0.8672 - val_accuracy: 0.7134
313/313 [==============================] - 1s 2ms/step - loss: 0.8672 - accuracy: 0.7
134
Test accuracy: 0.7134000062942505
```



In [9]:

```
# Create CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax'),  # 10 output classes for CIFAR-10
])

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

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

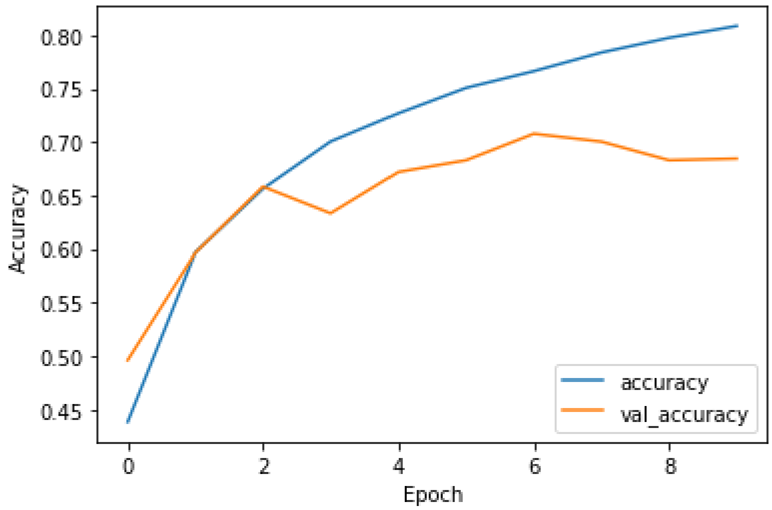
```
Epoch 1/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.0708 - accuracy: 0.2409 - val_loss: 1.8409 - val_accuracy: 0.3339
Epoch 2/10
1563/1563 [==============================] - 10s 7ms/step - loss: 1.6765 - accuracy: 0.3966 - val_loss: 1.5816 - val_accuracy: 0.4287
Epoch 3/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.4992 - accuracy: 0.4625 - val_loss: 1.5324 - val_accuracy: 0.4678
Epoch 4/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.3859 - accuracy: 0.5073 - val_loss: 1.3369 - val_accuracy: 0.5180
Epoch 5/10
1563/1563 [==============================] - 10s 7ms/step - loss: 1.2930 - accuracy: 0.5433 - val_loss: 1.2802 - val_accuracy: 0.5444
Epoch 6/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.2165 - accuracy: 0.5700 - val_loss: 1.2965 - val_accuracy: 0.5517
Epoch 7/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.1503 - accuracy: 0.5967 - val_loss: 1.1826 - val_accuracy: 0.5856
Epoch 8/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0928 - accuracy: 0.6181 - val_loss: 1.1001 - val_accuracy: 0.6100
Epoch 9/10
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0438 - accuracy: 0.6347 - val_loss: 1.1159 - val_accuracy: 0.5987
Epoch 10/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.9988 - accuracy: 0.6533 - val_loss: 1.0537 - val_accuracy: 0.6319
313/313 [==============================] - 1s 2ms/step - loss: 1.0537 - accuracy: 0.6319
Test accuracy: 0.6319000124931335
```



In [10]:
```
# Create CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
```

```python
    layers.Dense(10, activation='softmax'),  # 10 output classes for CIFAR-10
])

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

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/10
1563/1563 [==============================] - 11s 7ms/step - loss: 1.5557 - accuracy:
0.4384 - val_loss: 1.3610 - val_accuracy: 0.4962
Epoch 2/10
1563/1563 [==============================] - 10s 7ms/step - loss: 1.1467 - accuracy:
0.5968 - val_loss: 1.1247 - val_accuracy: 0.5964
Epoch 3/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.9815 - accuracy:
0.6563 - val_loss: 0.9868 - val_accuracy: 0.6583
Epoch 4/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.8702 - accuracy:
0.7005 - val_loss: 1.0705 - val_accuracy: 0.6333
Epoch 5/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.7896 - accuracy:
0.7270 - val_loss: 0.9581 - val_accuracy: 0.6720
Epoch 6/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.7225 - accuracy:
0.7508 - val_loss: 0.9885 - val_accuracy: 0.6830
Epoch 7/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.6723 - accuracy:
0.7664 - val_loss: 0.9011 - val_accuracy: 0.7078
Epoch 8/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.6267 - accuracy:
0.7839 - val_loss: 0.9414 - val_accuracy: 0.7004
Epoch 9/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.5884 - accuracy:
0.7976 - val_loss: 1.1393 - val_accuracy: 0.6830
Epoch 10/10
1563/1563 [==============================] - 10s 7ms/step - loss: 0.5589 - accuracy:
0.8087 - val_loss: 1.0724 - val_accuracy: 0.6845
313/313 [==============================] - 1s 2ms/step - loss: 1.0724 - accuracy: 0.6
845
Test accuracy: 0.684499979019165
```



Results

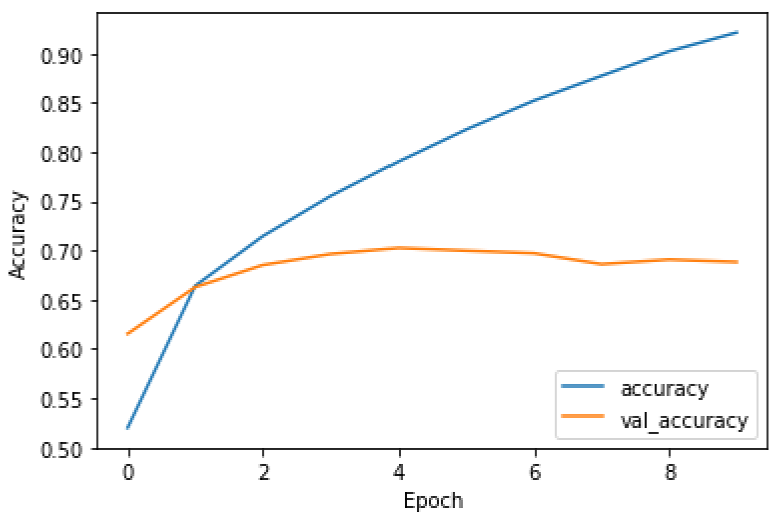
Upon evaluating the performance of optimizers in a CNN model, the Adam optimizer emerges as the most effective among the three—Adam, SGD, and RMSProp—utilized in the image classification task, achieving an accuracy score of 71.34%.

Layers

In [11]:

```
# Create a Shallow Networks model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax'),  # 10 output classes for CIFAR-10
])

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

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

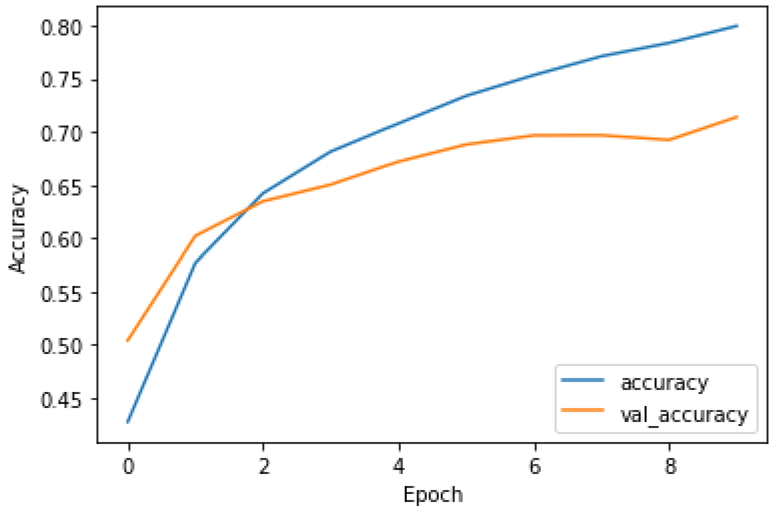
```
Epoch 1/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.3470 - accuracy:
0.5200 - val_loss: 1.0885 - val_accuracy: 0.6154
Epoch 2/10
1563/1563 [==============================] - 13s 9ms/step - loss: 0.9663 - accuracy:
0.6641 - val_loss: 0.9634 - val_accuracy: 0.6628
Epoch 3/10
1563/1563 [==============================] - 13s 9ms/step - loss: 0.8200 - accuracy:
0.7145 - val_loss: 0.9028 - val_accuracy: 0.6852
Epoch 4/10
1563/1563 [==============================] - 13s 9ms/step - loss: 0.7072 - accuracy:
0.7552 - val_loss: 0.8980 - val_accuracy: 0.6964
Epoch 5/10
1563/1563 [==============================] - 13s 9ms/step - loss: 0.6020 - accuracy:
0.7904 - val_loss: 0.9076 - val_accuracy: 0.7026
Epoch 6/10
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5072 - accuracy:
0.8230 - val_loss: 0.9314 - val_accuracy: 0.6998
Epoch 7/10
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4254 - accuracy:
0.8522 - val_loss: 1.0397 - val_accuracy: 0.6972
Epoch 8/10
1563/1563 [==============================] - 13s 9ms/step - loss: 0.3474 - accuracy:
0.8774 - val_loss: 1.1506 - val_accuracy: 0.6861
Epoch 9/10
1563/1563 [==============================] - 13s 9ms/step - loss: 0.2802 - accuracy:
0.9022 - val_loss: 1.2322 - val_accuracy: 0.6908
Epoch 10/10
1563/1563 [==============================] - 13s 8ms/step - loss: 0.2268 - accuracy:
0.9213 - val_loss: 1.3502 - val_accuracy: 0.6883
313/313 [==============================] - 1s 2ms/step - loss: 1.3502 - accuracy: 0.6
883
Test accuracy: 0.6883000135421753
```



In [22]:
```python
# Create a Deep Networks model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu'),

    layers.Flatten(),
```

```python
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax'),  # 10 output classes for CIFAR-10
])

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

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.5578 - accuracy:
0.4273 - val_loss: 1.3352 - val_accuracy: 0.5039
Epoch 2/10
1563/1563 [==============================] - 12s 8ms/step - loss: 1.1828 - accuracy:
0.5768 - val_loss: 1.1015 - val_accuracy: 0.6020
Epoch 3/10
1563/1563 [==============================] - 12s 8ms/step - loss: 1.0156 - accuracy:
0.6420 - val_loss: 1.0412 - val_accuracy: 0.6345
Epoch 4/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.9093 - accuracy:
0.6813 - val_loss: 0.9930 - val_accuracy: 0.6502
Epoch 5/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.8264 - accuracy:
0.7078 - val_loss: 0.9513 - val_accuracy: 0.6719
Epoch 6/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.7620 - accuracy:
0.7338 - val_loss: 0.8997 - val_accuracy: 0.6880
Epoch 7/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.7004 - accuracy:
0.7533 - val_loss: 0.8904 - val_accuracy: 0.6966
Epoch 8/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.6504 - accuracy:
0.7711 - val_loss: 0.8894 - val_accuracy: 0.6967
Epoch 9/10
1563/1563 [==============================] - 12s 7ms/step - loss: 0.6077 - accuracy:
0.7836 - val_loss: 0.9343 - val_accuracy: 0.6923
Epoch 10/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.5636 - accuracy:
0.7996 - val_loss: 0.8934 - val_accuracy: 0.7138
313/313 [==============================] - 1s 3ms/step - loss: 0.8934 - accuracy: 0.7
138
Test accuracy: 0.7138000130653381
```



Results

In this comparison, the deeper neural network model exhibited superior accuracy compared to the shallow network model. This aligns with the commonly observed trend in neural network architectures, highlighting the benefits of utilizing deeper structures. Deeper architectures excel in learning intricate patterns from data, resulting in enhanced learning capabilities and improved performance.