dxgit3zpu

January 24, 2025

Import libraries

[2]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras import datasets,models,layers
```

Libraries are imported for use in the project.

TensorFlow is used for deep learning models,

matplotlib is used for visualization, numpy for mathematical operations and pandas for data manipulation.

Data Preparation

Load CIFAR-10 dataset

[3]:
```
data = tf.keras.datasets.cifar10
```

[4]:
```
(train_images, train_labels), (test_images, test_labels) = data.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071             15s
0us/step
```

Loading the CIFAR-10 dataset from TensorFlow.

The dataset consists of 60,000 small (32x32) color images of 10 different classes.

'train_images' and 'train_labels' are used for the training set, and 'test_images' and 'test_labels' are used for the test set.

Inspect dataset shapes

[5]:
```
train_images.shape
```

[5]:
```
(50000, 32, 32, 3)
```

[6]:
```
test_images.shape
```

[6]:
```
(10000, 32, 32, 3)
```

[7]:
```
print(train_labels[0])
```

```
[6]
```

[8]:
```
print(train_images[1])
```

```
[[[154 177 187]
  [126 137 136]
  [105 104  95]
  …
  [ 91  95  71]
  [ 87  90  71]
  [ 79  81  70]]

 [[140 160 169]
  [145 153 154]
  [125 125 118]
  …
  [ 96  99  78]
  [ 77  80  62]
  [ 71  73  61]]

 [[140 155 164]
  [139 146 149]
  [115 115 112]
  …
  [ 79  82  64]
  [ 68  70  55]
  [ 67  69  55]]

 …

 [[175 167 166]
  [156 154 160]
  [154 160 170]
  …
  [ 42  34  36]
  [ 61  53  57]
  [ 93  83  91]]
```

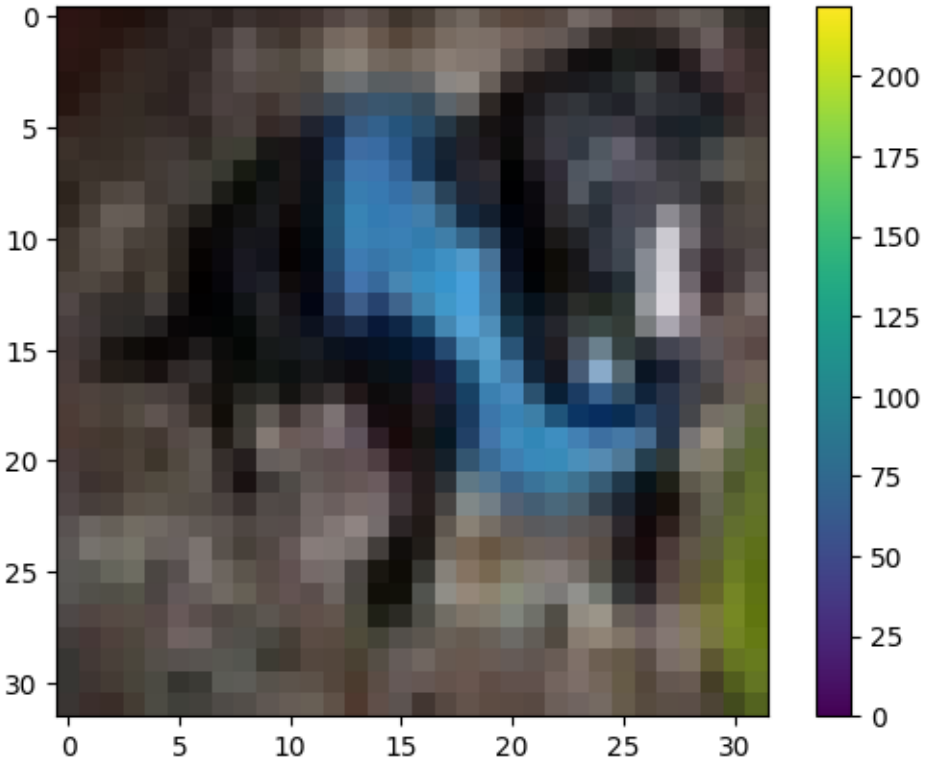
```
 [[165 154 128]
  [156 152 130]
  [159 161 142]
  …
  [103  93  96]
  [123 114 120]
  [131 121 131]]

 [[163 148 120]
  [158 148 122]
  [163 156 133]
  …
  [143 133 139]
  [143 134 142]
  [143 133 144]]]
```

The shapes of the training and test data sets are checked.

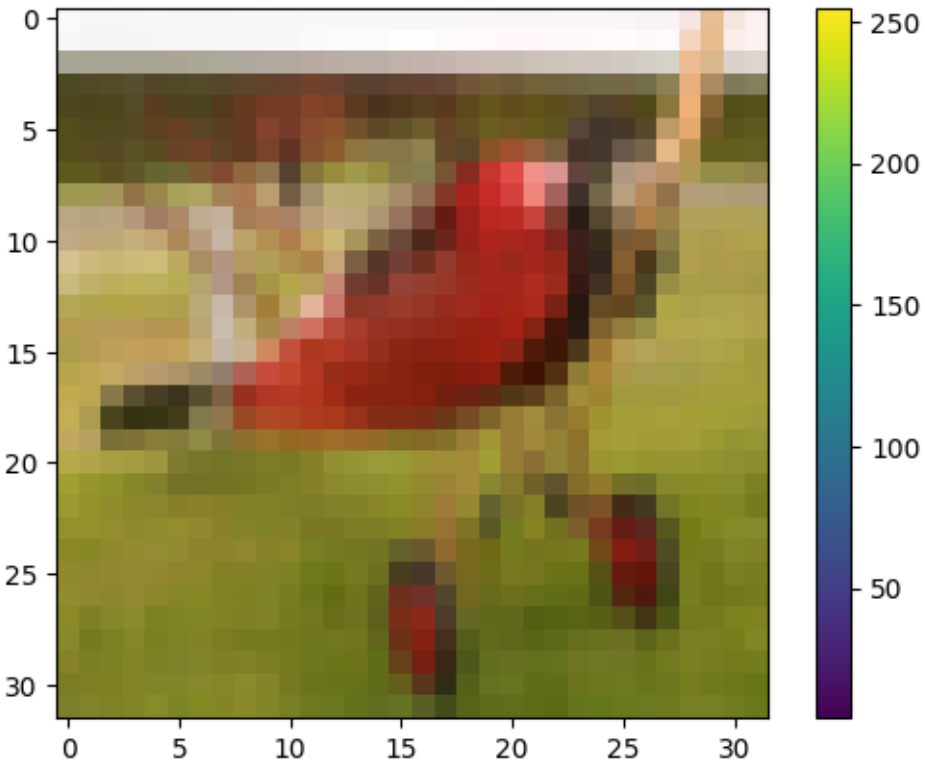
This helps us understand the dimensions and layout of the dataset.

Visualize a single image and its label

[9]:
```
plt.figure()
plt.imshow(train_images[27])
plt.colorbar()
plt.grid(False)

plt.show()
```



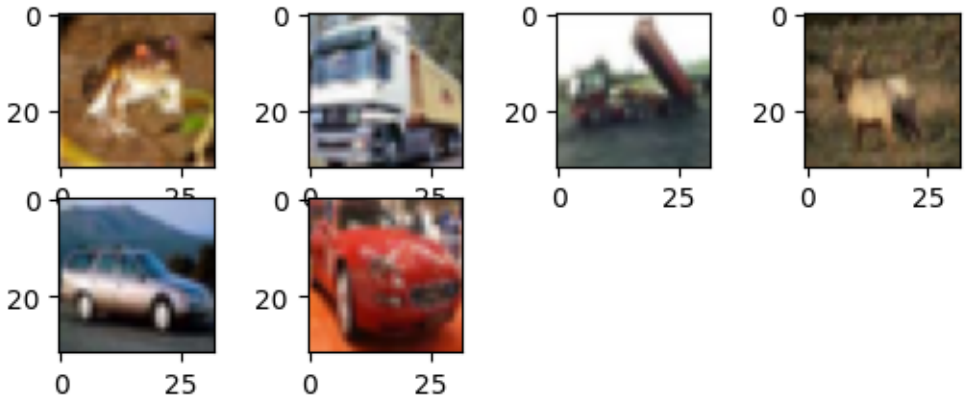
[10]:
```
plt.figure()
plt.imshow(train_images[35])
plt.colorbar()
plt.grid(False)

plt.show()
```



A single image and label from the training set is visualized.

This way, we understand how the images in the dataset look.

[15]:
```
for i in range(6):
        # define subplot
        plt.subplot(440 + 1 + i)
        # plot raw pixel data
        plt.imshow(train_images[i])
plt.show()
```



Normalize image pixel values to range [0, 1]

[16]:
```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

The pixel values in the image are normalized between 0 and 1.

This process allows the model to learn faster and more accurately.

Define Model 1: Simple CNN model

[17]:
```
model = tf.keras.models.Sequential([
                        tf.keras.layers.Conv2D(32, (3, 3), activation='relu',␣
 ↪input_shape=(32, 32, 3)),
                        tf.keras.layers.MaxPooling2D(2, 2),
                        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
                        tf.keras.layers.MaxPooling2D(2, 2),
                        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
                        tf.keras.layers.Flatten(),
                        tf.keras.layers.Dense(128, activation='relu'),
                        tf.keras.layers.Dense(10, activation='softmax')])
```

```
C:\Users\GHOST-V3\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

First model is being created.

This model has a simple Convolutional Neural Network (CNN) structure:

- First layer: A Convolutional layer with 32 filters, uses 3x3 filters and applies ReLU activation.

- First MaxPooling layer: Used to reduce the image size, with 2x2 size.

- Second Convolutional and MaxPooling layers: Creates deepened feature maps with more filters (64).

- Third Convolutional layer: Provides more depth but no pooling.

- Flatten layer: Converts multidimensional data to one-dimensional.

- Dense layers:

- The first Dense layer consists of 128 neurons and uses ReLU activation.

- The second Dense layer applies a softmax activation corresponding to 10 classes (CIFAR-10 classes).

---

[18]:
```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 128)	131,200
dense_1 (Dense)	(None, 10)	1,290

```
Total params: 188,810 (737.54 KB)

Trainable params: 188,810 (737.54 KB)

Non-trainable params: 0 (0.00 B)
```

The structure of the model is summarized.

This output allows us to see the total number of layers in the model, the size of the parameters, and the operations at each layer.

Compile the model

[19]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=20,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1563/1563             5s 3ms/step -
accuracy: 0.3715 - loss: 1.7064 - val_accuracy: 0.5371 - val_loss: 1.2899
Epoch 2/20
1563/1563             4s 3ms/step -
accuracy: 0.5890 - loss: 1.1604 - val_accuracy: 0.6367 - val_loss: 1.0256
Epoch 3/20
1563/1563             4s 3ms/step -
accuracy: 0.6571 - loss: 0.9713 - val_accuracy: 0.6513 - val_loss: 0.9817
Epoch 4/20
1563/1563             4s 3ms/step -
accuracy: 0.6929 - loss: 0.8626 - val_accuracy: 0.6794 - val_loss: 0.9143
Epoch 5/20
1563/1563             4s 3ms/step -
accuracy: 0.7217 - loss: 0.7899 - val_accuracy: 0.6919 - val_loss: 0.8824
Epoch 6/20
1563/1563             4s 3ms/step -
accuracy: 0.7493 - loss: 0.7091 - val_accuracy: 0.6965 - val_loss: 0.8769
Epoch 7/20
1563/1563             4s 3ms/step -
accuracy: 0.7687 - loss: 0.6586 - val_accuracy: 0.7125 - val_loss: 0.8531
Epoch 8/20
1563/1563             4s 3ms/step -
accuracy: 0.7822 - loss: 0.6119 - val_accuracy: 0.7162 - val_loss: 0.8547
Epoch 9/20
1563/1563             4s 3ms/step -
accuracy: 0.8064 - loss: 0.5531 - val_accuracy: 0.7158 - val_loss: 0.8599
Epoch 10/20
1563/1563             4s 3ms/step -
accuracy: 0.8206 - loss: 0.5061 - val_accuracy: 0.7076 - val_loss: 0.9020
Epoch 11/20
1563/1563             4s 3ms/step -
accuracy: 0.8392 - loss: 0.4610 - val_accuracy: 0.7186 - val_loss: 0.9073
Epoch 12/20
1563/1563             4s 3ms/step -
accuracy: 0.8534 - loss: 0.4176 - val_accuracy: 0.7152 - val_loss: 0.9556
Epoch 13/20
1563/1563             5s 3ms/step -
```

```
accuracy: 0.8649 - loss: 0.3810 - val_accuracy: 0.7096 - val_loss: 1.0002
Epoch 14/20
1563/1563             4s 3ms/step -
accuracy: 0.8813 - loss: 0.3409 - val_accuracy: 0.7128 - val_loss: 1.0400
Epoch 15/20
1563/1563             4s 3ms/step -
accuracy: 0.8895 - loss: 0.3086 - val_accuracy: 0.7113 - val_loss: 1.1073
Epoch 16/20
1563/1563             4s 3ms/step -
accuracy: 0.9026 - loss: 0.2763 - val_accuracy: 0.7159 - val_loss: 1.1227
Epoch 17/20
1563/1563             4s 3ms/step -
accuracy: 0.9144 - loss: 0.2451 - val_accuracy: 0.6963 - val_loss: 1.2518
Epoch 18/20
1563/1563             4s 3ms/step -
accuracy: 0.9206 - loss: 0.2241 - val_accuracy: 0.7044 - val_loss: 1.3217
Epoch 19/20
1563/1563             4s 3ms/step -
accuracy: 0.9275 - loss: 0.2046 - val_accuracy: 0.6979 - val_loss: 1.4145
Epoch 20/20
1563/1563             4s 3ms/step -
accuracy: 0.9337 - loss: 0.1887 - val_accuracy: 0.7056 - val_loss: 1.4247
```

Compiling the model.

- Optimization algorithm: 'adam' is used. This algorithm automatically adjusts the learning rate.

- Loss function: 'sparse_categorical_crossentropy' is suitable for multi-class classification problems where labels are encoded as integers.

- Performance metric: 'accuracy' is used to measure the accuracy of the model.

The model is trained for 20 epochs.

- Training set: 'train_images' and 'train_labels' are used.

- Validation set: 'test_images' and 'test_labels' are reserved for validation.

- Epoch: A training cycle. The model processes the dataset once in each epoch.

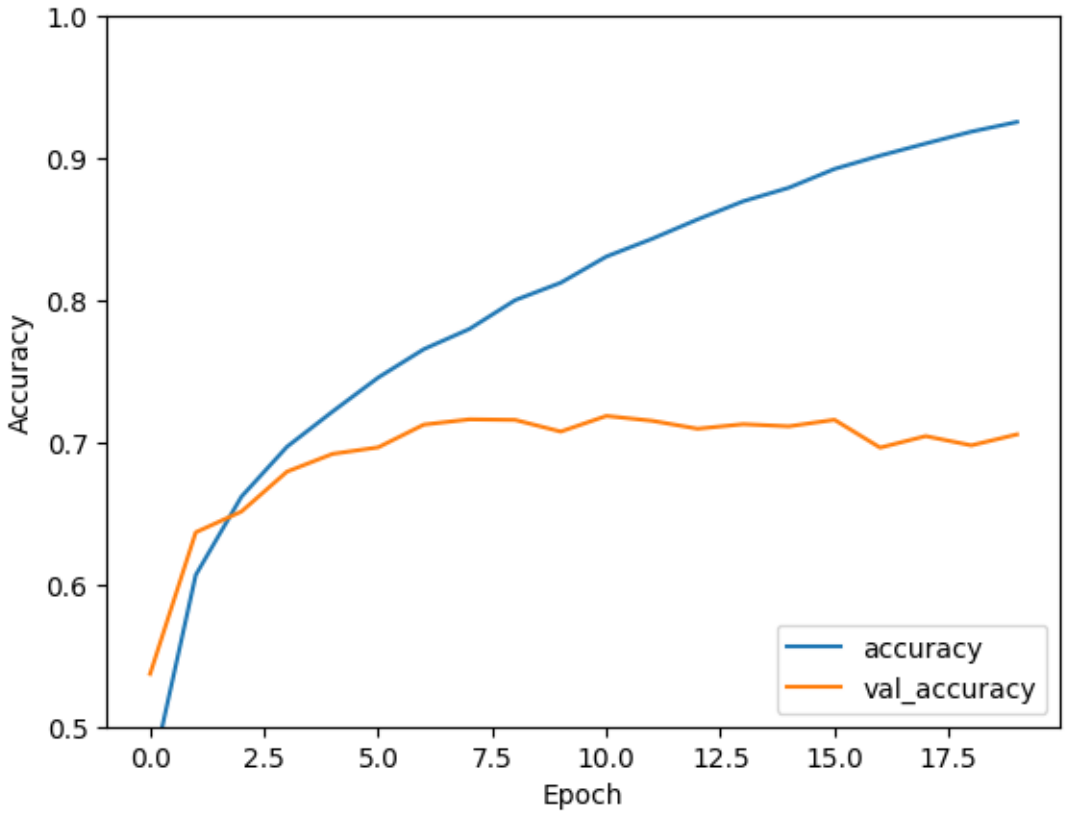
During training, the model's accuracy and loss values are recorded for each epoch.

---

[21]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

[21]:
```
<matplotlib.legend.Legend at 0x1fc35a8d110>
```



Training and validation accuracies are visualized.

- This graph shows how the model learns and improves over time.

- Training accuracy: Accuracy on the training set.

- Validation accuracy: Accuracy on the validation set.

Define Model 2: Expanded CNN model

[22]:
```
vgg_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32,␣
↪3), padding='same'),
```

```
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Creating a more complex model (Model 2):

- First two Convolutional layers: Depth is increased with 32 filters.

- First MaxPooling layer: Used to reduce the image size.

- Second group of Convolutional layers: More features are extracted with 64 filters.

- Second MaxPooling layer: Provides more dimension reduction.

- Dense layer: Learning capacity is increased with 256 neurons.

- Dropout layer: Overfitting is prevented by disabling 50% of random neurons.

- Last layer: Output is produced using softmax for 10 classes.

Compile VGG-like model

[23]:
```
vgg_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])

vgg_history = vgg_model.fit(train_images, train_labels, epochs=20,
                            validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1563/1563             57s 36ms/step -
accuracy: 0.3346 - loss: 1.7870 - val_accuracy: 0.5895 - val_loss: 1.1538
Epoch 2/20
1563/1563             56s 36ms/step -
accuracy: 0.6125 - loss: 1.0907 - val_accuracy: 0.6806 - val_loss: 0.9021
Epoch 3/20
```

```
1563/1563            56s 36ms/step -
accuracy: 0.7054 - loss: 0.8356 - val_accuracy: 0.7267 - val_loss: 0.8050
Epoch 4/20
1563/1563            56s 36ms/step -
accuracy: 0.7571 - loss: 0.6942 - val_accuracy: 0.7210 - val_loss: 0.8447
Epoch 5/20
1563/1563            56s 36ms/step -
accuracy: 0.7984 - loss: 0.5757 - val_accuracy: 0.7537 - val_loss: 0.7286
Epoch 6/20
1563/1563            56s 36ms/step -
accuracy: 0.8274 - loss: 0.4894 - val_accuracy: 0.7667 - val_loss: 0.7106
Epoch 7/20
1563/1563            59s 38ms/step -
accuracy: 0.8485 - loss: 0.4245 - val_accuracy: 0.7666 - val_loss: 0.7413
Epoch 8/20
1563/1563            59s 38ms/step -
accuracy: 0.8652 - loss: 0.3828 - val_accuracy: 0.7793 - val_loss: 0.7170
Epoch 9/20
1563/1563            58s 37ms/step -
accuracy: 0.8869 - loss: 0.3282 - val_accuracy: 0.7688 - val_loss: 0.8600
Epoch 10/20
1563/1563            59s 37ms/step -
accuracy: 0.8911 - loss: 0.3142 - val_accuracy: 0.7787 - val_loss: 0.9075
Epoch 11/20
1563/1563            58s 37ms/step -
accuracy: 0.9042 - loss: 0.2813 - val_accuracy: 0.7729 - val_loss: 0.9188
Epoch 12/20
1563/1563            58s 37ms/step -
accuracy: 0.9131 - loss: 0.2503 - val_accuracy: 0.7657 - val_loss: 0.9122
Epoch 13/20
1563/1563            59s 38ms/step -
accuracy: 0.9145 - loss: 0.2483 - val_accuracy: 0.7737 - val_loss: 0.9608
Epoch 14/20
1563/1563            59s 38ms/step -
accuracy: 0.9154 - loss: 0.2491 - val_accuracy: 0.7773 - val_loss: 0.9075
Epoch 15/20
1563/1563            57s 36ms/step -
accuracy: 0.9227 - loss: 0.2254 - val_accuracy: 0.7694 - val_loss: 0.9946
Epoch 16/20
1563/1563            57s 36ms/step -
accuracy: 0.9252 - loss: 0.2252 - val_accuracy: 0.7691 - val_loss: 1.0532
Epoch 17/20
1563/1563            56s 36ms/step -
accuracy: 0.9286 - loss: 0.2157 - val_accuracy: 0.7815 - val_loss: 1.0207
Epoch 18/20
1563/1563            58s 37ms/step -
accuracy: 0.9336 - loss: 0.1927 - val_accuracy: 0.7661 - val_loss: 1.0634
Epoch 19/20
```

```
1563/1563              59s 37ms/step -
accuracy: 0.9284 - loss: 0.2186 - val_accuracy: 0.7726 - val_loss: 1.1483
Epoch 20/20
1563/1563              60s 38ms/step -
accuracy: 0.9331 - loss: 0.2009 - val_accuracy: 0.7645 - val_loss: 1.1576
```

Training a VGG-like model:

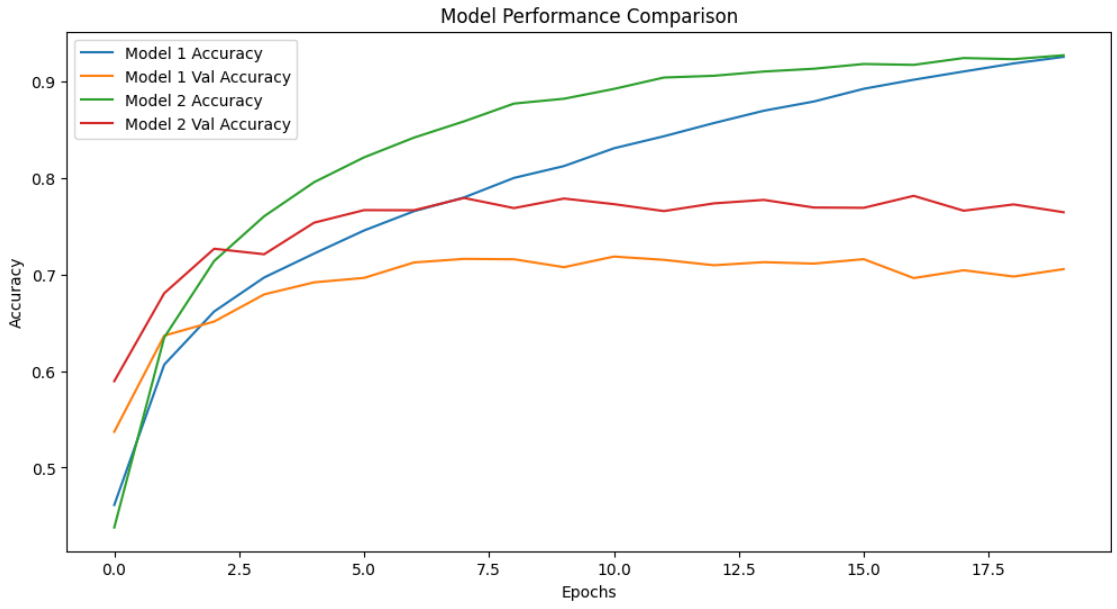
- Training is done for 20 epochs.

- Performance is recorded for training and validation sets.

This model is expected to have deeper layers and provide higher accuracy, but training time may also be longer.

---

[24]:
```
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Model 1 Accuracy')
plt.plot(history.history['val_accuracy'], label='Model 1 Val Accuracy')
plt.plot(vgg_history.history['accuracy'], label='Model 2 Accuracy')
plt.plot(vgg_history.history['val_accuracy'], label='Model 2 Val Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Performance Comparison')
plt.legend()
plt.show()
```



Comparing the performance of the models:

- The training and validation accuracies of Model 1 (simpler CNN model) and Model 2 (VGG-like model) are shown in the same graph.

- The accuracy of both models increases during training, but the validation accuracy is important to understand the generalization ability of the model.

- If the validation accuracy of Model 2 is higher, it can be concluded that the more complex structure increases the generalization capacity.

This graph helps us visually understand which model is better.

[ ]: