Model1:

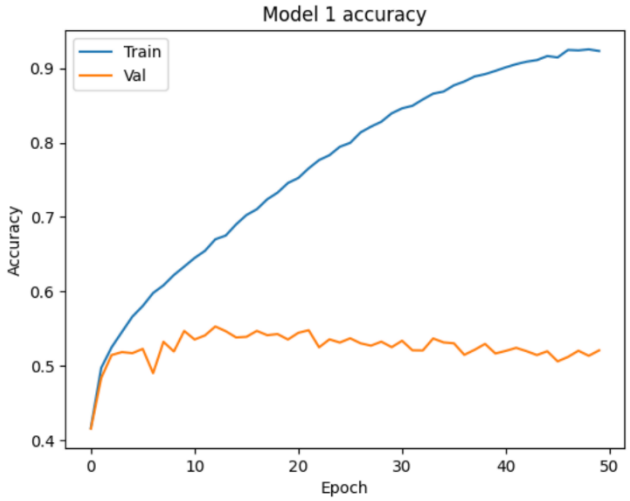
Number of Parameters, MAC Operations and Output Size for Model 1

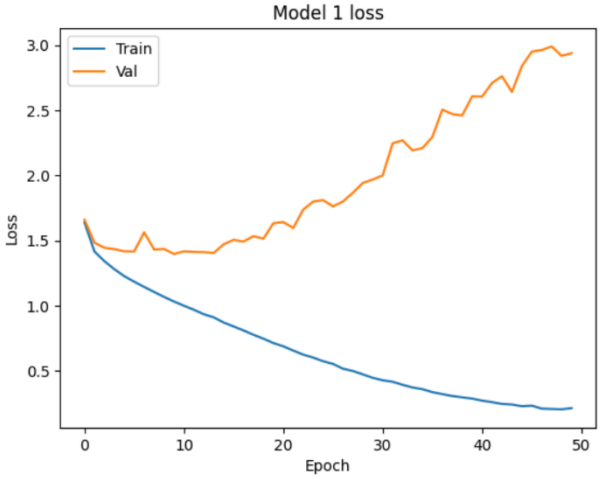
Layer Name	Output Size	Parameters	MAC Operations
Conv2D	16x16x32	896	295,014,912 (16x16x32x3x3x3x32)
BatchNormalization	16x16x32	256	0
Conv2D	8x8x64	18,496	118,111,488 (8x8x64x3x3x32x64)
BatchNormalization	8x8x64	512	0
Conv2D	4x4x128	73,856	590,252,544 (4x4x128x3x3x64x128)
BatchNormalization	4x4x128	512	0
Conv2D	4x4x128	1,47,584	590,252,544 (4x4x128x3x3x128x128)
BatchNormalization	4x4x128	512	0
Conv2D	4x4x128	1,47,584	590,252,544 (4x4x128x3x3x128x128)
BatchNormalization	4x4x128	512	0
Conv2D	4x4x128	1,47,584	590,252,544 (4x4x128x3x3x128x128)
BatchNormalization	4x4x128	512	0
MaxPooling2D	1x1x128	0	0
Flatten	128	0	0
Dense	128	16,512	16,512
BatchNormalization	128	512	0
Dense	10	1,290	1,280

Formula for Calculating MACs (output_height * output_width * output_channels) * (kernel_height * kernel_width * input_channels) * output_channels

Did you observe any overfitting? Should the model train for longer, shorter, or about that number of epochs.

The Validation accuracy is lower than the Training accuracy. The Validation loss is higher than the Training loss. Hence, Model1 is Overfitting.





Does it correctly label the picture?

Sample image of Dog is tested on Model 1 and it correctly classified

```
uploaded = files.upload()
img = Image.open("dog.jpg") # Loading Image
img = img.resize((32, 32)) # Resizing
x = np.array(img) # Convert the image to a numpy array
x = np.expand_dims(x, axis=0)
model = keras.models.load_model("model1.h5")
# Make a prediction on the image
pred = model.predict(x)
# Get the predicted class label
class_index= np.argmax(pred)
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
class_name = class_names[class_index]
print(class_name)
```

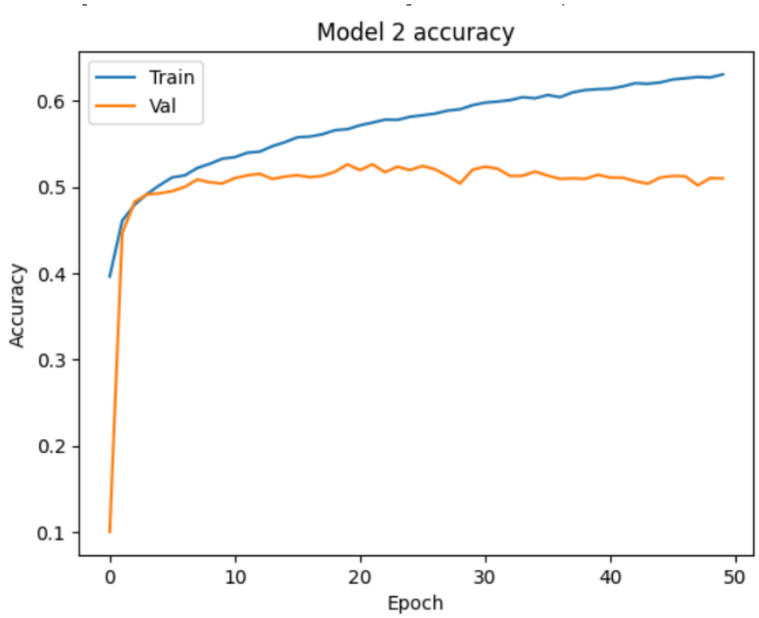
```
Choose Files  dog.jpg
• dog.jpg(image/jpeg) - 884038 bytes, last modified: 2/7/2024 - 100% done
Saving dog.jpg to dog.jpg
1/1 [==============================] - 0s 192ms/step
dog
```

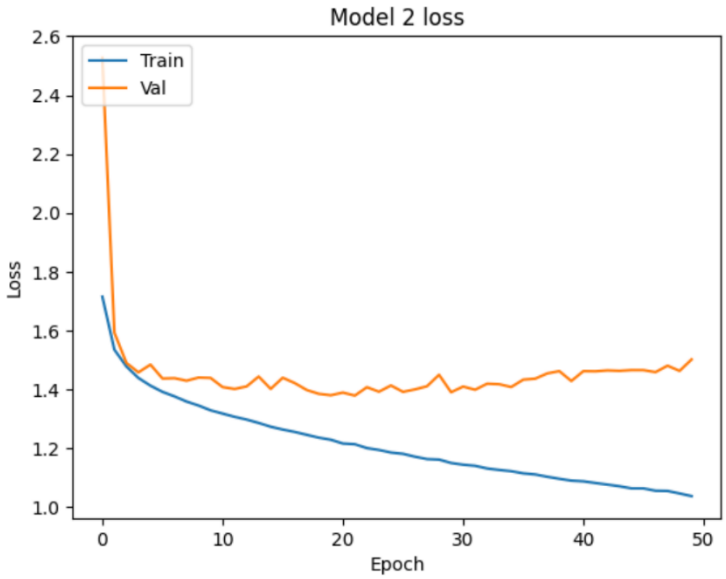
Model2

Layer	Output Shape	Parameters	MAC Operations
Conv2D	(16, 16, 32)	896	294,912 (16x16x32x3x3x3x32)
BatchNormalization	(16, 16, 32)	128	0
DepthwiseConv2D	(8, 8, 32)	288	57,600 (8x8x32x3x3x1)
Conv2D	(8, 8, 64)	2,112	36,864 (8x8x64x1x1x32)
BatchNormalization	(8, 8, 64)	256	0
DepthwiseConv2D	(4, 4, 64)	576	14,400 (4x4x64x3x3x1)
Conv2D	(4, 4, 128)	8,320	9,216 (4x4x128x1x1x64)
BatchNormalization	(4, 4, 128)	512	0
DepthwiseConv2D	(2, 2, 128)	1,152	3,600 (2x2x128x3x3x1)
Conv2D	(2, 2, 128)	16,512	512 (2x2x128x1x1x128)
BatchNormalization	(2, 2, 128)	512	0
DepthwiseConv2D	(1, 1, 128)	1,152	3,600 (1x1x128x3x3x1)
Conv2D	(1, 1, 128)	16,512	128 (1x1x128x1x1x128)
BatchNormalization	(1, 1, 128)	512	0
DepthwiseConv2D	(1, 1, 128)	1,152	3,600 (1x1x128x3x3x1)
Conv2D	(1, 1, 128)	16,512	128 (1x1x128x1x1x128)
BatchNormalization	(1, 1, 128)	512	0
GlobalAveragePooling2D	(None, 128)	0	0
Flatten	(None, 128)	0	0
Dense	(None, 128)	16,512	16,512
BatchNormalization	(None, 128)	512	0
Dense	(None, 10)	1,290	1,280

Did you observe any overfitting? Should the model train for longer, shorter, or about that number of epochs.

The validation accuracy is lower than the training accuracy. The validation loss is higher that the training loss. So, this model is overfitting.





Does it correctly label the picture?

Sample image of Dog is tested on Model 2 and it correctly classified.

```
img = Image.open("dog.jpg") # Loading Image
img = img.resize((32, 32)) # Resizing
x = np.array(img) # Convert the image to a numpy array
x = np.expand_dims(x, axis=0)
model = keras.models.load_model("model2.h5")
# Make a prediction on the image
pred = model.predict(x)
# Get the predicted class label
class_index= np.argmax(pred)
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
class_name = class_names[class_index]
print(class_name)
```

```
1/1 [==============================] - 0s 196ms/step
dog
```

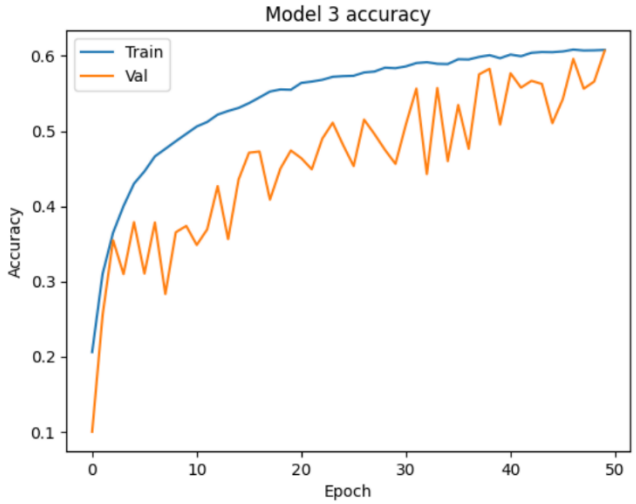
Model3

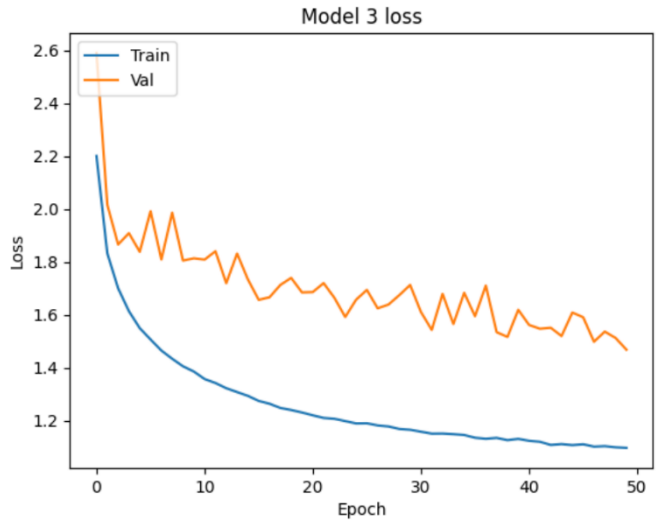
Layer	Output Shape	Parameters	MAC Operations
InputLayer	(None, 32, 32, 3)	0	0
Conv2D	(None, 16, 16, 32)	896	294,912 (16x16x32x3x3x3x32)
BatchNormalization	(None, 16, 16, 32)	128	0
Dropout	(None, 16, 16, 32)	0	0
Conv2D	(None, 8, 8, 64)	18,496	36,864 (8x8x64x3x3x32)
BatchNormalization	(None, 8, 8, 64)	256	0
Dropout	(None, 8, 8, 64)	0	0
Conv2D	(None, 4, 4, 128)	73,856	36,864 (4x4x128x3x3x64)
BatchNormalization	(None, 4, 4, 128)	512	0
Dropout	(None, 4, 4, 128)	0	0
Conv2D	(None, 4, 4, 128)	4,224	11,520 (4x4x128x1x1x32)
Add	(None, 4, 4, 128)	0	0
Dropout	(None, 4, 4, 128)	0	0
Conv2D	(None, 2, 2, 128)	1,47,584	11,520 (2x2x128x3x3x1)
BatchNormalization	(None, 2, 2, 128)	512	0
Dropout	(None, 2, 2, 128)	0	0
Conv2D	(None, 1, 1, 128)	1,47,584	32 (1x1x128x1x1x128)
BatchNormalization	(None, 1, 1, 128)	512	0
Dropout	(None, 1, 1, 128)	0	0
Add	(None, 4, 4, 128)	0	0
Dropout	(None, 4, 4, 128)	0	0
Conv2D	(None, 4, 4, 128)	1,47,584	36,864 (4x4x128x3x3x32)
BatchNormalization	(None, 4, 4, 128)	512	0
Dropout	(None, 4, 4, 128)	0	0
Conv2D	(None, 4, 4, 128)	1,47,584	36,864 (4x4x128x3x3x32)
BatchNormalization	(None, 4, 4, 128)	512	0
Dropout	(None, 4, 4, 128)	0	0
Add	(None, 4, 4, 128)	0	0
Dropout	(None, 4, 4, 128)	0	0
MaxPooling2D	(None, 1, 1, 128)	0	0

Flatten	(None, 128)	0	0
Dense	(None, 128)	16,512	16,512
BatchNormalization	(None, 128)	512	0
Dense	(None, 10)	1,290	1,280

Did you observe any overfitting? Should the model train for longer, shorter, or about that number of epochs.

The validation accuracy is almost similar to the training accuracy. The validation loss is nearly same as training loss. So, this model is slightly overfitting.





Does it correctly label the picture?

A picture of an dog was tested, and Model3 Classified it as Truck.

```
img = Image.open("dog.jpg") # Loading Image
img = img.resize((32, 32)) # Resizing
x = np.array(img) # Convert the image to a numpy array
x = np.expand_dims(x, axis=0)
model = keras.models.load_model("model3.h5")
# Make a prediction on the image
pred = model.predict(x)
# Get the predicted class label
class_index= np.argmax(pred)
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
class_name = class_names[class_index]
print(class_name)
```

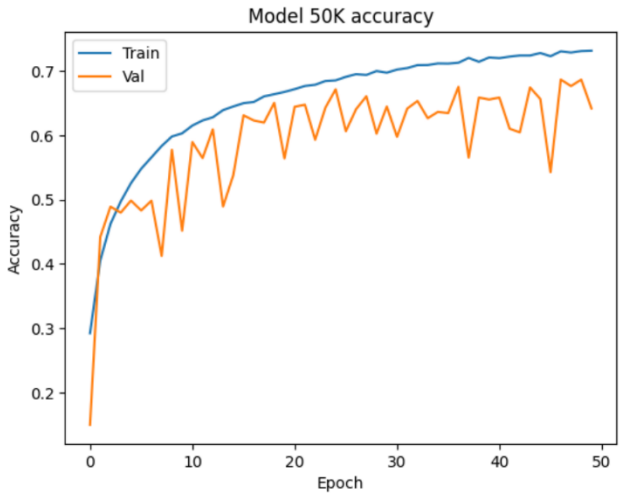
```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function
1/1 [==============================] - 0s 209ms/step
truck
```

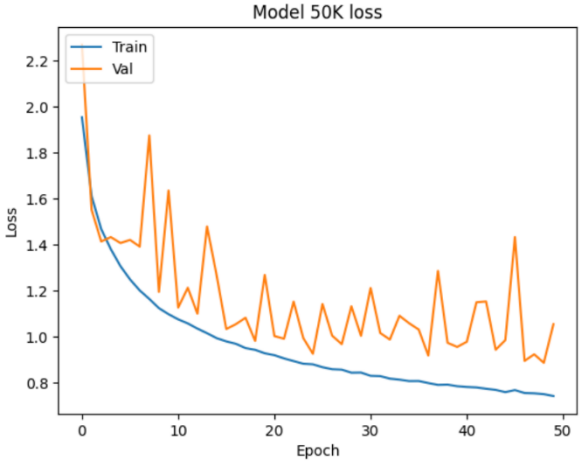
Model_50K

Layer	Output Shape	Parameters	MAC Operations
Conv2D	(None, 16, 16, 32)	416	819,200 (16x16x32x3x3x32)
BatchNormalization	(None, 16, 16, 32)	128	0
Activation	(None, 16, 16, 32)	0	0
SeparableConv2D	(None, 8, 8, 64)	2,400	36,864 (8x8x64x3x3x32)
BatchNormalization	(None, 8, 8, 64)	256	0
SeparableConv2D	(None, 4, 4, 128)	8,576	36,864 (4x4x128x3x3x64)
BatchNormalization	(None, 4, 4, 128)	512	0
Dropout	(None, 4, 4, 128)	0	0
SeparableConv2D	(None, 4, 4, 128)	17,664	36,864 (4x4x128x3x3x128)
BatchNormalization	(None, 4, 4, 128)	512	0
Dropout	(None, 4, 4, 128)	0	0
MaxPooling2D	(None, 1, 1, 128)	0	0
Flatten	(None, 128)	0	0
Dense	(None, 128)	16,512	16,512
Dropout	(None, 128)	0	0
Dense	(None, 10)	1,290	1,280

Did you observe any overfitting? Should the model train for longer, shorter, or about that number of epochs.

The validation accuracy is similar to the training accuracy. The validation loss is also the same as training loss. Model_50K performing better than previous models





Does it correctly label the picture?

A picture of a dog was tested, and it classified it as cat.

```
img = Image.open("dog.jpg") # Loading Image
img = img.resize((32, 32)) # Resizing
x = np.array(img) # Convert the image to a numpy array
x = np.expand_dims(x, axis=0)
model = keras.models.load_model("best_model.h5")
# Make a prediction on the image
pred = model.predict(x)
# Get the predicted class label
class_index= np.argmax(pred)
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
class_name = class_names[class_index]
print(class_name)
```

```
1/1 [==============================] - 0s 374ms/step
cat
```

How did the three models compare? Consider final accuracy, time per epoch to train, number of epochs needed to reach a given accuracy, overfitting.

The model50K exhibited the highest accuracy, reaching 0.742. The average training time per epoch ranged from 4 to 5 seconds. Training the models for 50 epochs proved effective, with overfitting showing improvement from model 1 to model50K.

What did you observe? What model architecture or training decisions made the most difference?

Reducing the number of model layers has had an impact on accuracy. As model layers are reduced, accuracy increases and overfitting decreases. Additionally, this reduction decreases the number of parameters.