ICP- 8

V V S Murthy Kolla
700729142

Video link:
https://drive.google.com/file/d/1kVIOiSPWbZsA1mRnFyQ46nXVrcth9Y-7/view?usp=share_link

Github link: https://github.com/murthykolla/ICP-8.git

LeNet Model:

- For hyperparameter tuning, I have included the learning rate for the Adam optimization.
- Also increased the number of epochs to 50.
- I have also used the early stopping method and monitored the validation accuracy improvement with patience 5.
- Plotted the accuracy and loss plot using the history with subplots method for all the models.
- Plotted the confusion matrix heat map for all the models.
- We can see an increase in the test and validation accuracy from the base model.
- Used batch size of 64 to increase the model's accuracy in the .fit method.

AlexNet Model:

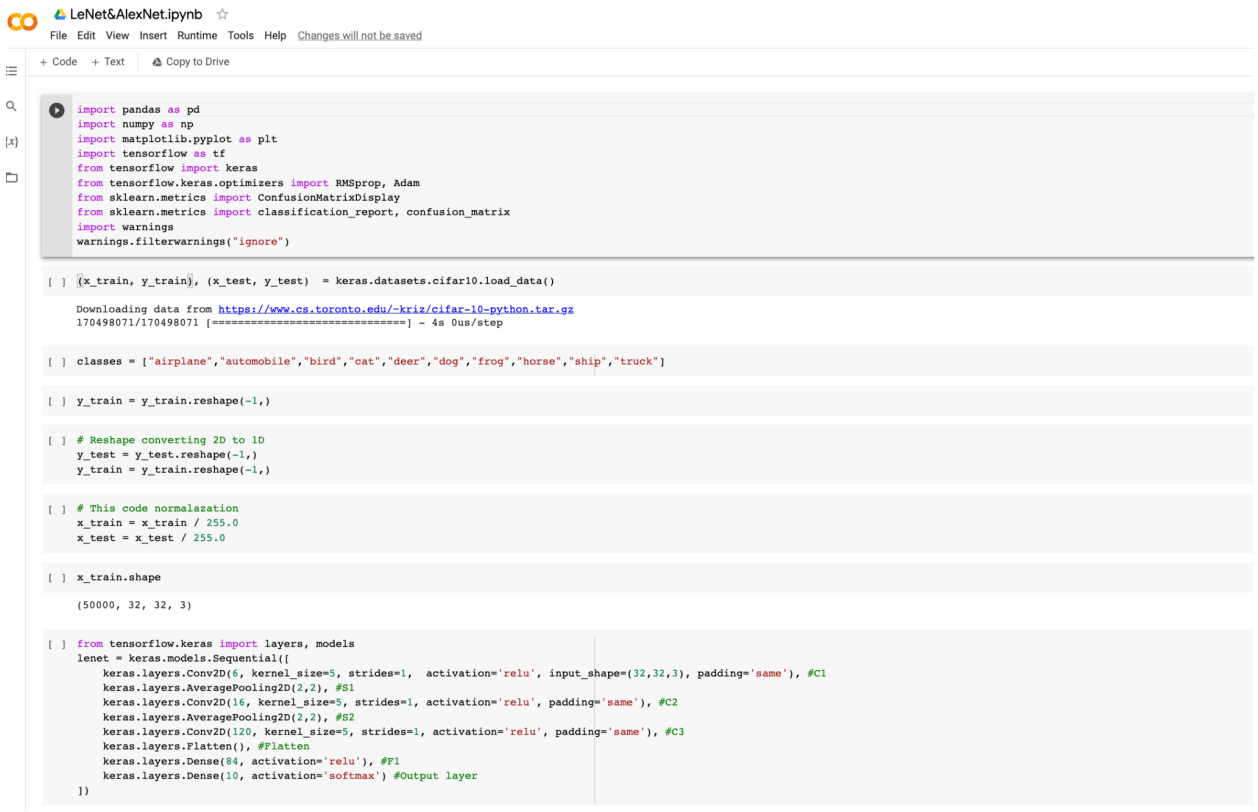
- For hyperparameter tuning, I have increased the dropout rate to 0.5. So that the model can learn from unknown neurons during forward and backpropagation.
- Included the learning rate of 0.001 and Adam optimizer to compile the model.
- Increased the kernel size of the first Conv2D layer.
- I have also used the early stopping method and monitored the validation accuracy improvement with patience 5.
- Used the batch size of 32, which better suits the model, to get better accuracy results.
- Plotted the accuracy and loss plot using the history with subplots method for all the models.
- Plotted the confusion matrix heat map for all the models.
- Here, we can see an increase in the validation accuracy and test accuracy score of 63.95%.
- For data visualization, I tested the prediction on a random image and showed the corresponding image.

VGG16 Model:

- Here, I have used the SGD optimizer with parameters of learning rate of 0.01, decay of 1e-6, momentum 0,9, and Nesterov as True.
- Then I called this optimizer into the compile method to get better accuracy results with SGD optimizer.
- Changed the number of epochs to 25 and batch size to 128 for parameter tuning for better results.
- Then, I saved the model to a .h5 file and plotted the accuracy and loss graphs.
- Plotted the heatmap of the confusion matrix.
- For data visualization, I tested the prediction on a random image and showed the corresponding image.
- Finally, I have loaded the saved .h5 file to predict the test image and printed the prediction class and expected class.

VGG19

- Increased the number of epochs to 100 and batch size to 64.
- Changed the model architecture with the Cifar100 dataset.
- Increased the dense layers to 4096 and the last dense layer to 100.
- Included the dropout layer with a drop rate of 0.5.
- Used the model.fit method with a batch size of 128 and a number of epochs as 25.
- Plotted the accuracy and loss plots.
- Printed the testing accuracy of the model and plotted the confusion accuracy.



LeNet&AlexNet.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers import RMSprop, Adam
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")
```

```
[ ] (x_train, y_train), (x_test, y_test)  = keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

```
[ ] classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

```
[ ] y_train = y_train.reshape(-1,)
```

```
[ ] # Reshape converting 2D to 1D
    y_test = y_test.reshape(-1,)
    y_train = y_train.reshape(-1,)
```

```
[ ] # This code normalazation
    x_train = x_train / 255.0
    x_test = x_test / 255.0
```

```
[ ] x_train.shape
```

```
(50000, 32, 32, 3)
```

```
[ ] from tensorflow.keras import layers, models
    lenet = keras.models.Sequential([
        keras.layers.Conv2D(6, kernel_size=5, strides=1,  activation='relu', input_shape=(32,32,3), padding='same'), #C1
        keras.layers.AveragePooling2D(2,2), #S1
        keras.layers.Conv2D(16, kernel_size=5, strides=1, activation='relu', padding='same'), #C2
        keras.layers.AveragePooling2D(2,2), #S2
        keras.layers.Conv2D(120, kernel_size=5, strides=1, activation='relu', padding='same'), #C3
        keras.layers.Flatten(), #Flatten
        keras.layers.Dense(84, activation='relu'), #F1
        keras.layers.Dense(10, activation='softmax') #Output layer
    ])
```

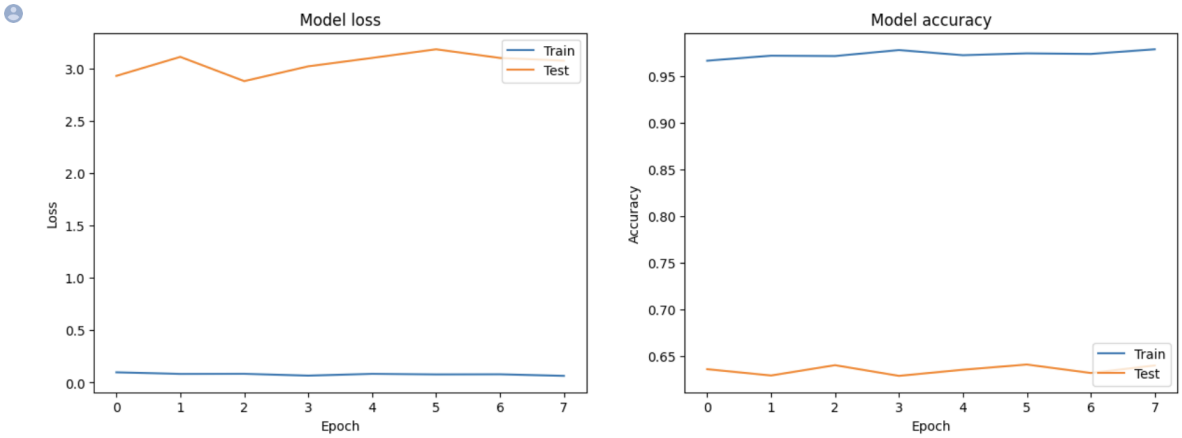
```
[ ] lenet.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 32, 32, 6)         456

 average_pooling2d_6 (Averag  (None, 16, 16, 6)        0
 ePooling2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 16)        2416

 average_pooling2d_7 (Averag  (None, 8, 8, 16)         0
 ePooling2D)

 conv2d_11 (Conv2D)          (None, 8, 8, 120)         48120

 flatten_3 (Flatten)         (None, 7680)              0

 dense_6 (Dense)             (None, 84)                645204

 dense_7 (Dense)             (None, 10)                850

=================================================================
Total params: 697,046
Trainable params: 697,046
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler

# define early stopping callback
earlystop = EarlyStopping(monitor='val_loss', patience=5)

# compile model
lenet.compile(optimizer=Adam(learning_rate=0.001), loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])

# fit model with callbacks
history = lenet.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), callbacks=[earlystop],verbose=1)
```

```
Epoch 1/50
782/782 [==============================] - 6s 5ms/step - loss: 0.0981 - accuracy: 0.9667 - val_loss: 2.9333 - val_accuracy: 0.6355
Epoch 2/50
782/782 [==============================] - 4s 4ms/step - loss: 0.0827 - accuracy: 0.9721 - val_loss: 3.1152 - val_accuracy: 0.6287
Epoch 3/50
782/782 [==============================] - 4s 5ms/step - loss: 0.0838 - accuracy: 0.9717 - val_loss: 2.8833 - val_accuracy: 0.6398
Epoch 4/50
782/782 [==============================] - 4s 5ms/step - loss: 0.0663 - accuracy: 0.9781 - val_loss: 3.0246 - val_accuracy: 0.6283
Epoch 5/50
782/782 [==============================] - 4s 5ms/step - loss: 0.0836 - accuracy: 0.9726 - val_loss: 3.1052 - val_accuracy: 0.6349
Epoch 6/50
782/782 [==============================] - 3s 4ms/step - loss: 0.0782 - accuracy: 0.9746 - val_loss: 3.1877 - val_accuracy: 0.6405
Epoch 7/50
782/782 [==============================] - 4s 5ms/step - loss: 0.0793 - accuracy: 0.9740 - val_loss: 3.1048 - val_accuracy: 0.6314
Epoch 8/50
782/782 [==============================] - 4s 5ms/step - loss: 0.0641 - accuracy: 0.9790 - val_loss: 3.0789 - val_accuracy: 0.6395
```

```
import matplotlib.pyplot as plt

# Plot the results
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# Plot training and validation accuracy
axs[1].plot(history.history['accuracy'])
axs[1].plot(history.history['val_accuracy'])
axs[1].set_title('Model accuracy')
axs[1].set_ylabel('Accuracy')
axs[1].set_xlabel('Epoch')
axs[1].legend(['Train', 'Test'], loc='lower right')

# Plot training and validation loss
axs[0].plot(history.history['loss'])
axs[0].plot(history.history['val_loss'])
axs[0].set_title('Model loss')
axs[0].set_ylabel('Loss')
axs[0].set_xlabel('Epoch')
axs[0].legend(['Train', 'Test'], loc='upper right')

plt.show()
```

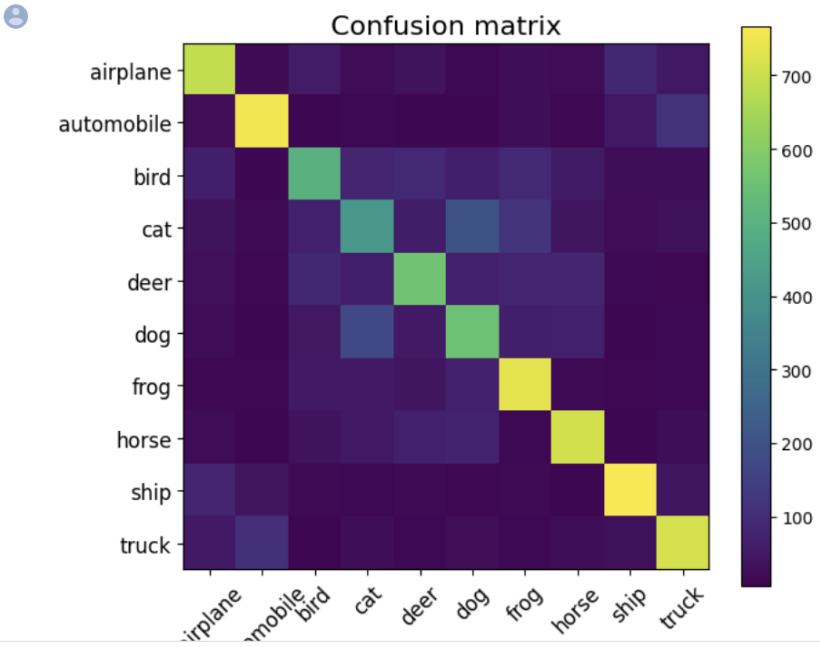



```
[ ] from sklearn.metrics import confusion_matrix
    from sklearn.metrics import ConfusionMatrixDisplay
    y_predictions= lenet.predict(x_test)
    y_predictions.reshape(-1,)
    y_predictions= np.argmax(y_predictions, axis=1)

    confusion_matrix(y_test, y_predictions)
```

```
[ ] from sklearn.metrics import confusion_matrix
    from sklearn.metrics import ConfusionMatrixDisplay
    y_predictions= lenet.predict(x_test)
    y_predictions.reshape(-1,)
    y_predictions= np.argmax(y_predictions, axis=1)

    confusion_matrix(y_test, y_predictions)
```

```
313/313 [==============================] - 1s 2ms/step
array([[690,  15,  57,  19,  34,  13,  20,  17,  86,  49],
       [ 25, 753,   6,  13,   7,   7,  21,   8,  47, 113],
       [ 67,   7, 501,  82,  92,  67,  89,  54,  20,  21],
       [ 34,  16,  69, 411,  60, 199, 118,  43,  19,  31],
       [ 26,   8,  88,  66, 560,  69,  79,  82,  12,  10],
       [ 17,   7,  46, 173,  52, 553,  65,  70,   5,  12],
       [  9,   8,  52,  50,  41,  68, 736,  14,  10,  12],
       [ 18,   6,  37,  47,  68,  76,  13, 710,   5,  20],
       [ 81,  38,  14,  13,  16,  10,  14,   5, 766,  43],
       [ 51, 109,   7,  21,  11,  24,  11,  20,  31, 715]])
```

```
# confusion matrix and accuracy
from sklearn.metrics import confusion_matrix, accuracy_score
plt.figure(figsize=(7, 6))
plt.title('Confusion matrix', fontsize=16)
plt.imshow(confusion_matrix(y_test, y_predictions))
plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```



```
[ ] print("Test accuracy:", accuracy_score(y_test, y_predictions))
```

```
Test accuracy: 0.6395
```

```
L = 8
W = 8
fig, axes = plt.subplots(L, W, figsize = (20,20))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i])
    axes[i].set_title("Predicted = {}\n Actual  = {}".format(classes[y_predictions[i]], classes[y_test[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```

```
 dropout_24 (Dropout)        (None, 4096)              0

 dense_45 (Dense)            (None, 4096)              16781312

 dropout_25 (Dropout)        (None, 4096)              0

 dense_46 (Dense)            (None, 10)                40970

=================================================================
Total params: 16,992,292
Trainable params: 16,992,292
Non-trainable params: 0
_________________________________________________________________
```

```python
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler

# define early stopping callback
earlystop = EarlyStopping(monitor='val_loss', patience=5)

# compile model
AlexNet.compile(optimizer=Adam(learning_rate=0.001), loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])

# fit model with callbacks
history = AlexNet.fit(x_train, y_train, batch_size=32, epochs=50, validation_data=(x_test, y_test), callbacks=[earlystop],verbose=1)
```

```
Epoch 1/50
1563/1563 [==============================] - 13s 6ms/step - loss: 2.0881 - accuracy: 0.1795 - val_loss: 1.8452 - val_accuracy: 0.2964
Epoch 2/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.7712 - accuracy: 0.3294 - val_loss: 1.6706 - val_accuracy: 0.3578
Epoch 3/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.6495 - accuracy: 0.3915 - val_loss: 1.5894 - val_accuracy: 0.4165
Epoch 4/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.5902 - accuracy: 0.4226 - val_loss: 1.5480 - val_accuracy: 0.4264
Epoch 5/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.5533 - accuracy: 0.4366 - val_loss: 1.5612 - val_accuracy: 0.4386
Epoch 6/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.5283 - accuracy: 0.4467 - val_loss: 1.5326 - val_accuracy: 0.4420
Epoch 7/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.5043 - accuracy: 0.4584 - val_loss: 1.5107 - val_accuracy: 0.4561
Epoch 8/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4799 - accuracy: 0.4726 - val_loss: 1.4988 - val_accuracy: 0.4596
Epoch 9/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4606 - accuracy: 0.4769 - val_loss: 1.4803 - val_accuracy: 0.4657
Epoch 10/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4381 - accuracy: 0.4872 - val_loss: 1.4988 - val_accuracy: 0.4687
Epoch 11/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4211 - accuracy: 0.4955 - val_loss: 1.4805 - val_accuracy: 0.4731
Epoch 12/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4070 - accuracy: 0.5003 - val_loss: 1.4811 - val_accuracy: 0.4651
Epoch 13/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3889 - accuracy: 0.5078 - val_loss: 1.4747 - val_accuracy: 0.4763
Epoch 14/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3821 - accuracy: 0.5123 - val_loss: 1.4708 - val_accuracy: 0.4830
Epoch 15/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3658 - accuracy: 0.5201 - val_loss: 1.5083 - val_accuracy: 0.4631
Epoch 16/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3583 - accuracy: 0.5221 - val_loss: 1.4588 - val_accuracy: 0.4778
Epoch 17/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3425 - accuracy: 0.5276 - val_loss: 1.4721 - val_accuracy: 0.4818
Epoch 18/50
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3301 - accuracy: 0.5328 - val_loss: 1.4761 - val_accuracy: 0.4769
```

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Conv2D
    from tensorflow.keras.layers import Dropout
    from tensorflow.keras.layers import Flatten
    from tensorflow.keras.optimizers import SGD, Adam
    from keras.layers.convolutional import Convolution2D
    from keras.layers.convolutional import MaxPooling2D
```

```
#Define Alexnet Model
AlexNet = Sequential()
AlexNet.add(Conv2D(filters=16,kernel_size=(11,11),strides=(4,4),input_shape=(32,32,3), activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Conv2D(60,(5,5),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(1,1)))
AlexNet.add(Conv2D(60,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(30,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(20,(3,3),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Flatten())
AlexNet.add(Dense(4096, activation='relu'))
AlexNet.add(Dropout(0.5))
AlexNet.add(Dense(4096, activation='relu'))
AlexNet.add(Dropout(0.5))
AlexNet.add(Dense(10,activation='softmax'))

AlexNet.compile(optimizer=Adam(learning_rate=0.001), loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
AlexNet.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_102 (Conv2D)         (None, 6, 6, 16)          5824

 max_pooling2d_54 (MaxPoolin  (None, 3, 3, 16)         0
 g2D)

 conv2d_103 (Conv2D)         (None, 3, 3, 60)          24060

 max_pooling2d_55 (MaxPoolin  (None, 2, 2, 60)         0
 g2D)

 conv2d_104 (Conv2D)         (None, 2, 2, 60)          32460

 conv2d_105 (Conv2D)         (None, 2, 2, 30)          16230

 conv2d_106 (Conv2D)         (None, 2, 2, 20)          5420

 max_pooling2d_56 (MaxPoolin  (None, 1, 1, 20)         0
 g2D)

 flatten_16 (Flatten)        (None, 20)                0

 dense_44 (Dense)            (None, 4096)              86016

 dropout_24 (Dropout)        (None, 4096)              0

 dense_45 (Dense)            (None, 4096)              16781312
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD, Adam
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D
```

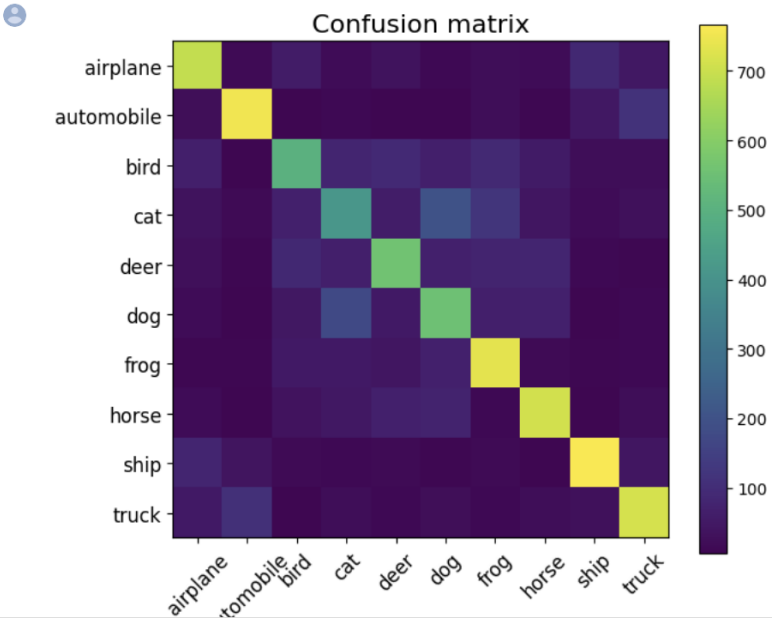
```
#Define Alexnet Model
AlexNet = Sequential()
AlexNet.add(Conv2D(filters=16,kernel_size=(11,11),strides=(4,4),input_shape=(32,32,3), activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Conv2D(60,(5,5),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(1,1)))
AlexNet.add(Conv2D(60,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(30,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(20,(3,3),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Flatten())
AlexNet.add(Dense(4096, activation='relu'))
AlexNet.add(Dropout(0.5))
AlexNet.add(Dense(4096, activation='relu'))
AlexNet.add(Dropout(0.5))
AlexNet.add(Dense(10,activation='softmax'))

AlexNet.compile(optimizer=Adam(learning_rate=0.001), loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
AlexNet.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_102 (Conv2D)          (None, 6, 6, 16)          5824

max_pooling2d_54 (MaxPoolin  (None, 3, 3, 16)          0
g2D)

conv2d_103 (Conv2D)          (None, 3, 3, 60)          24060

max_pooling2d_55 (MaxPoolin  (None, 2, 2, 60)          0
g2D)

conv2d_104 (Conv2D)          (None, 2, 2, 60)          32460

conv2d_105 (Conv2D)          (None, 2, 2, 30)          16230

conv2d_106 (Conv2D)          (None, 2, 2, 20)          5420

max_pooling2d_56 (MaxPoolin  (None, 1, 1, 20)          0
g2D)

flatten_16 (Flatten)         (None, 20)                0

dense_44 (Dense)             (None, 4096)              86016

dropout_24 (Dropout)         (None, 4096)              0

dense_45 (Dense)             (None, 4096)              16781312
```

```
[ ] from sklearn.metrics import confusion_matrix
    from sklearn.metrics import ConfusionMatrixDisplay
    y_predictions= lenet.predict(x_test)
    y_predictions.reshape(-1,)
    y_predictions= np.argmax(y_predictions, axis=1)

    confusion_matrix(y_test, y_predictions)
```

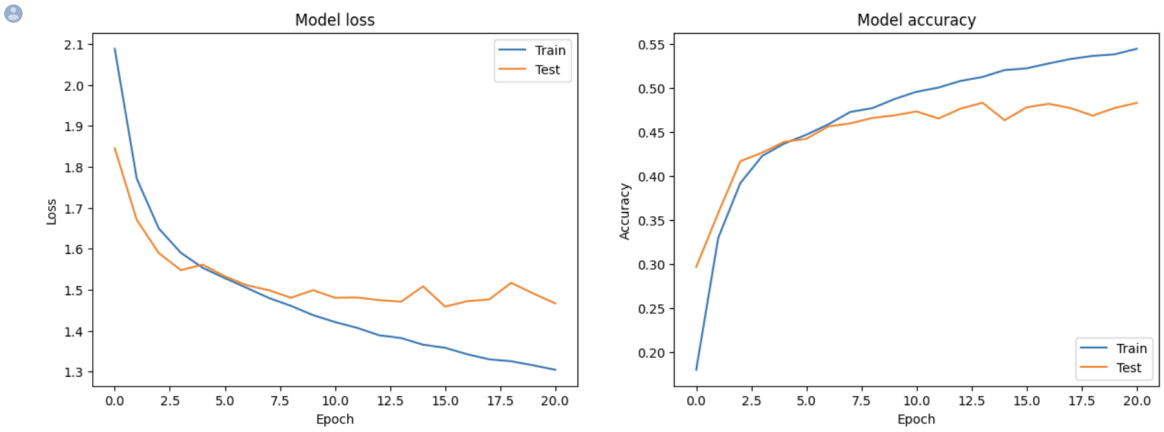
```
313/313 [==============================] - 1s 2ms/step
array([[690,  15,  57,  19,  34,  13,  20,  17,  86,  49],
       [ 25, 753,   6,  13,   7,   7,  21,   8,  47, 113],
       [ 67,   7, 501,  82,  92,  67,  89,  54,  20,  21],
       [ 34,  16,  69, 411,  60, 199, 118,  43,  19,  31],
       [ 26,   8,  88,  66, 560,  69,  79,  82,  12,  10],
       [ 17,   7,  46, 173,  52, 553,  65,  70,   5,  12],
       [  9,   8,  52,  50,  41,  68, 736,  14,  10,  12],
       [ 18,   6,  37,  47,  68,  76,  13, 710,   5,  20],
       [ 81,  38,  14,  13,  16,  10,  14,   5, 766,  43],
       [ 51, 109,   7,  21,  11,  24,  11,  20,  31, 715]])
```

```
# confusion matrix and accuracy
from sklearn.metrics import confusion_matrix, accuracy_score
plt.figure(figsize=(7, 6))
plt.title('Confusion matrix', fontsize=16)
plt.imshow(confusion_matrix(y_test, y_predictions))
plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```



```
[ ] Epoch 20/50
    1563/1563 [==============================] - 9s 6ms/step - loss: 1.3155 - accuracy: 0.5381 - val_loss: 1.4908 - val_accuracy: 0.4770
    Epoch 21/50
    1563/1563 [==============================] - 9s 6ms/step - loss: 1.3046 - accuracy: 0.5444 - val_loss: 1.4668 - val_accuracy: 0.4829
```

```python
import matplotlib.pyplot as plt

# Plot the results
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# Plot training and validation accuracy
axs[1].plot(history.history['accuracy'])
axs[1].plot(history.history['val_accuracy'])
axs[1].set_title('Model accuracy')
axs[1].set_ylabel('Accuracy')
axs[1].set_xlabel('Epoch')
axs[1].legend(['Train', 'Test'], loc='lower right')

# Plot training and validation loss
axs[0].plot(history.history['loss'])
axs[0].plot(history.history['val_loss'])
axs[0].set_title('Model loss')
axs[0].set_ylabel('Loss')
axs[0].set_xlabel('Epoch')
axs[0].legend(['Train', 'Test'], loc='upper right')

plt.show()
```

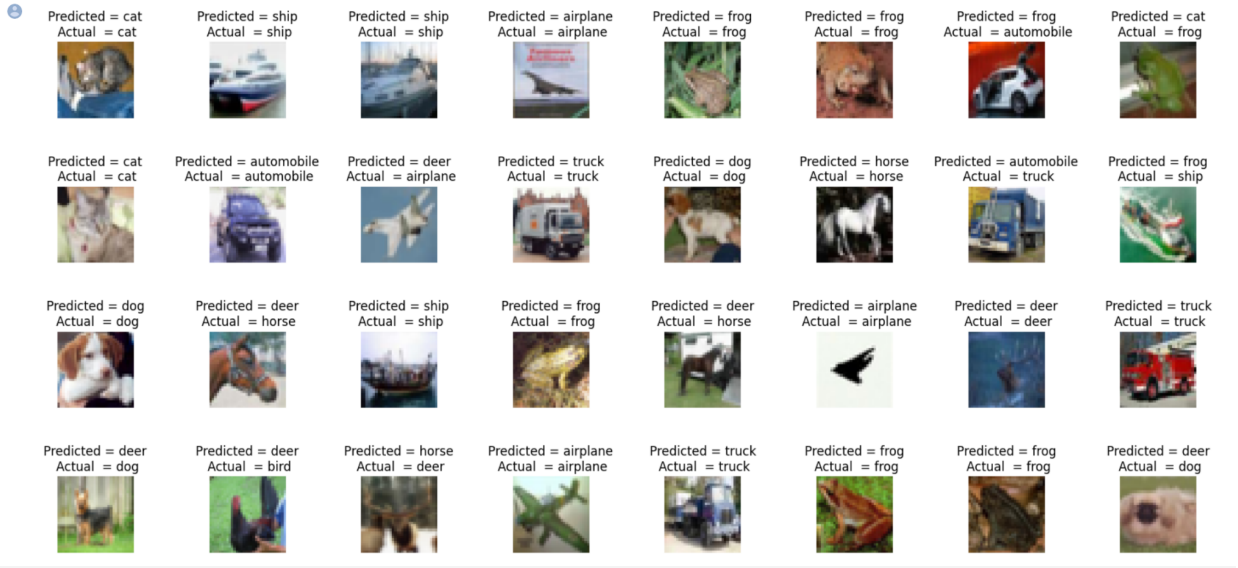



```
[ ] y_predictions1 = AlexNet.predict(x_test)
    y_predictions1.reshape(-1,)
```

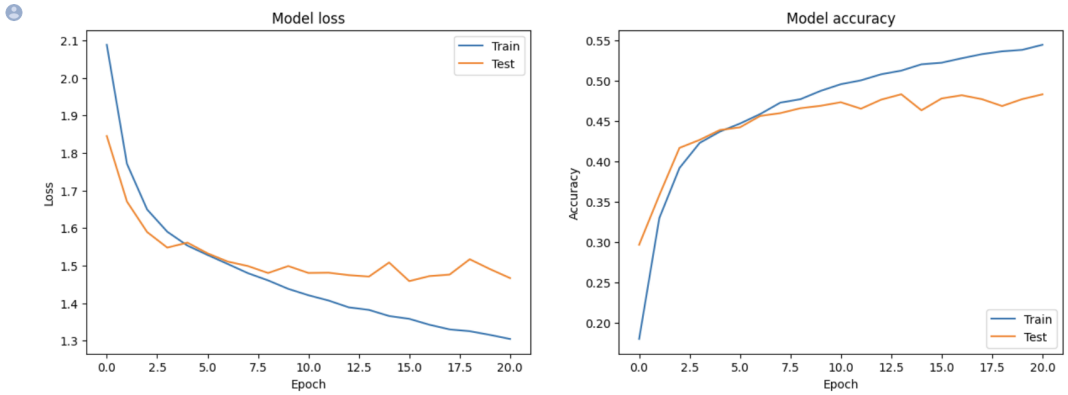
```
[ ] print("Test accuracy:", accuracy_score(y_test, y_predictions))
```

```
Test accuracy: 0.6395
```

```
L = 8
W = 8
fig, axes = plt.subplots(L, W, figsize = (20,20))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i])
    axes[i].set_title("Predicted = {}\n Actual  = {}".format(classes[y_predictions[i]], classes[y_test[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```



```
import matplotlib.pyplot as plt

# Plot the results
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# Plot training and validation accuracy
axs[1].plot(history.history['accuracy'])
axs[1].plot(history.history['val_accuracy'])
axs[1].set_title('Model accuracy')
axs[1].set_ylabel('Accuracy')
axs[1].set_xlabel('Epoch')
axs[1].legend(['Train', 'Test'], loc='lower right')

# Plot training and validation loss
axs[0].plot(history.history['loss'])
axs[0].plot(history.history['val_loss'])
axs[0].set_title('Model loss')
axs[0].set_ylabel('Loss')
axs[0].set_xlabel('Epoch')
axs[0].legend(['Train', 'Test'], loc='upper right')

plt.show()
```

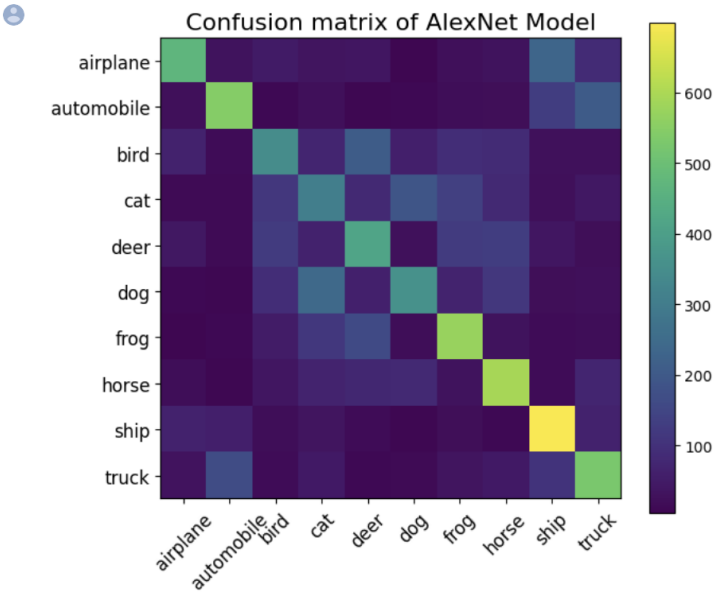



```
[ ] y_predictions1 = AlexNet.predict(x_test)
    y_predictions1.reshape(-1,)
    y_predictions1= np.argmax(y_predictions1, axis=1)

    confusion_matrix(y_test, y_predictions1)

    313/313 [==============================] - 1s 2ms/step
    array([[468,  30,  50,  36,  38,   5,  24,  29, 234,  86],
```

```
[ ] y_predictions1 = AlexNet.predict(x_test)
    y_predictions1.reshape(-1,)
    y_predictions1= np.argmax(y_predictions1, axis=1)

    confusion_matrix(y_test, y_predictions1)
```

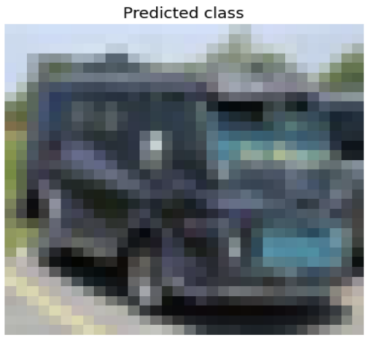
```
313/313 [==============================] - 1s 2ms/step
array([[468,  30,  50,  36,  38,   5,  24,  29, 234,  86],
       [ 23, 545,  10,  24,   9,  10,  20,  22, 130, 207],
       [ 64,  16, 348,  73, 208,  60,  92,  87,  26,  26],
       [ 13,  13, 113, 305,  81, 191, 136,  81,  25,  42],
       [ 41,  13, 125,  65, 415,  28, 125, 130,  37,  21],
       [ 11,   8,  92, 243,  61, 360,  67, 113,  21,  24],
       [  4,  10,  52, 116, 160,  20, 571,  31,  16,  20],
       [ 18,   9,  38,  68,  75,  80,  29, 594,  15,  74],
       [ 66,  60,  18,  36,  16,   9,  21,  11, 698,  65],
       [ 33, 166,  16,  46,  11,  13,  36,  47, 107, 525]])
```

```
# confusion matrix and accuracy
plt.figure(figsize=(7, 6))
plt.title('Confusion matrix of AlexNet Model', fontsize=16)
plt.imshow(confusion_matrix(y_test, y_predictions1))
plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```



```python
# Define the class names for CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Choose a random image from the test set
index = np.random.randint(0, x_test.shape[0])
image = x_test[index]

# Make a prediction on the image using the pre-trained model
prediction = AlexNet.predict(np.expand_dims(image, axis=0))

class_index = tf.argmax(prediction, axis=1)
class_name = tf.keras.utils.to_categorical(class_index, num_classes=10)
actual_class = y_test[index]

# Visualize the image and the predicted and actual classes
plt.imshow(image)
plt.title(f"Predicted class")
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 24ms/step
```



VGG16.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

```
[ ] import keras
    from keras.models import Sequential
    from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D
    from keras.datasets import cifar10
    from keras import optimizers
    from keras.optimizers import SGD
    from matplotlib import pyplot as plt
```

```
[ ] # generate cifar10 data
    (x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

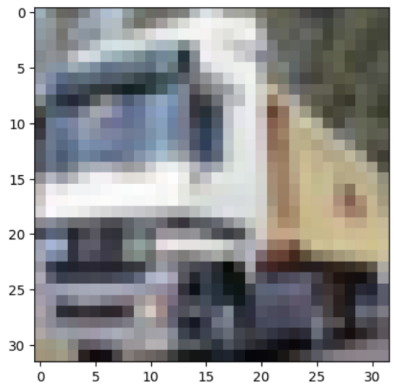
```
[ ] # config parameters
    num_classes = 10
    input_shape = x_train.shape[1:4]
    # Compile the model
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
```

```
    # convert label to one-hot
    one_hot_y_train = keras.utils.to_categorical(y_train,num_classes=num_classes)
    one_hot_y_test = keras.utils.to_categorical(y_test,num_classes=num_classes)
```

+ Code + Text

```
[ ] # check data
    plt.imshow(x_train[1])
    print(x_train[1].shape)
```

```
(32, 32, 3)
```




```
[ ] # build model(similar to VGG16, only change the input and output shape)
    model = Sequential()
    model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
    model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
```

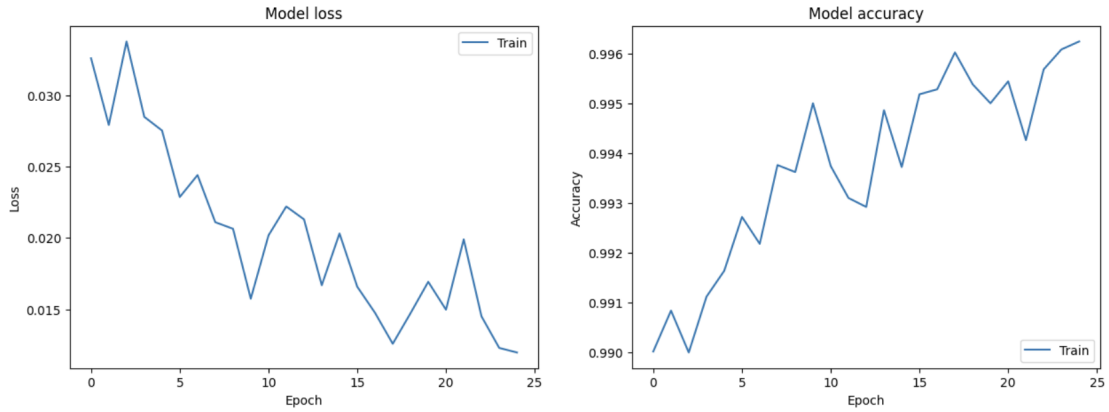
```
# build model(similar to VGG16, only change the input and output shape)
model = Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(4096,activation='relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
[ ]
model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# check model
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_52 (Conv2D)          (None, 32, 32, 64)        1792

 conv2d_53 (Conv2D)          (None, 32, 32, 64)        36928

 max_pooling2d_20 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_54 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_55 (Conv2D)          (None, 16, 16, 128)       147584

 max_pooling2d_21 (MaxPoolin  (None, 8, 8, 128)        0
 g2D)

 conv2d_56 (Conv2D)          (None, 8, 8, 256)         295168

 conv2d_57 (Conv2D)          (None, 8, 8, 256)         590080

 conv2d_58 (Conv2D)          (None, 8, 8, 256)         590080
```

```python
import matplotlib.pyplot as plt

# Plot the results
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# Plot training and validation accuracy
axs[1].plot(history.history['accuracy'])
axs[1].set_title('Model accuracy')
axs[1].set_ylabel('Accuracy')
axs[1].set_xlabel('Epoch')
axs[1].legend(['Train', 'Test'], loc='lower right')

# Plot training and validation loss
axs[0].plot(history.history['loss'])
axs[0].set_title('Model loss')
axs[0].set_ylabel('Loss')
axs[0].set_xlabel('Epoch')
axs[0].legend(['Train', 'Test'], loc='upper right')

plt.show()
```



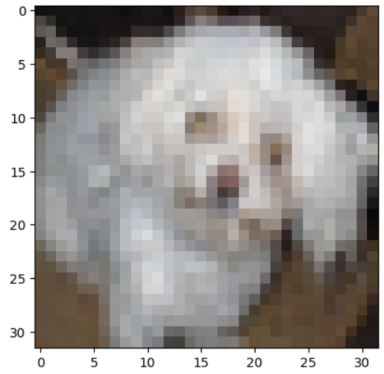

```python
[ ] import numpy as np
    y_predictions1 = model.predict(x_test)
    y_predictions1.reshape(-1,)
```

```
[ ] # evaluate
    print(model.metrics_names)
    model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
['loss', 'accuracy']
20/20 [==============================] - 0s 8ms/step - loss: 1.5592 - accuracy: 0.7990
[1.5591590404510498, 0.7990000247955322]
```

```
# predict
plt.imshow(x_test[1000])

result = model.predict(x_test[1000:1001]).tolist()
predict = 0
expect = y_test[1000][0]
for i,_ in enumerate(result[0]):
    if result[0][i] > result[0][predict]:
        predict = i
print("predict class:",predict)
print("expected class:",expect)
```

```
1/1 [==============================] - 0s 28ms/step
predict class: 5
expected class: 5
```

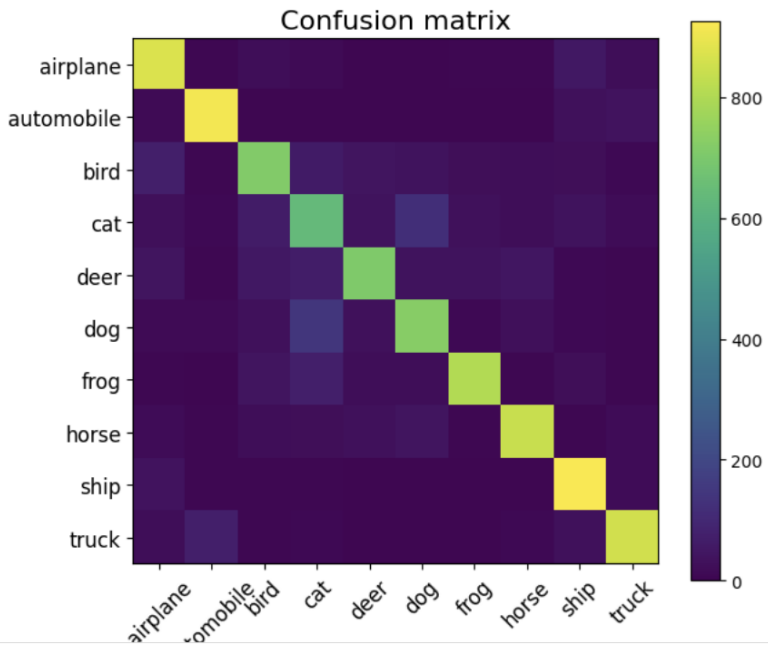



```
[ ] # save model
    model.save("keras-VGG16-cifar10.h5")
```

```
[ ] import numpy as np
    y_predictions1 = model.predict(x_test)
    y_predictions1.reshape(-1,)
    y_predictions1= np.argmax(y_predictions1, axis=1)
    from sklearn.metrics import classification_report, confusion_matrix
    confusion_matrix(y_test, y_predictions1)
```

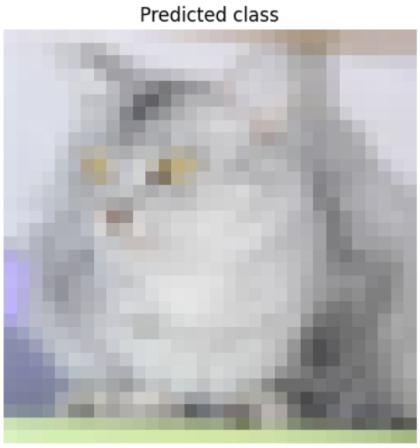
```
313/313 [==============================] - 1s 3ms/step
array([[874,   7,  20,  11,   2,   2,   5,   6,  53,  20],
       [ 12, 914,   1,   0,   0,   1,   1,   0,  32,  39],
       [ 71,   5, 714,  58,  41,  35,  22,  19,  25,  10],
       [ 28,  10,  65, 636,  36, 122,  30,  21,  35,  17],
       [ 42,   7,  51,  66, 706,  34,  35,  44,  10,   5],
       [ 12,  11,  32, 141,  31, 723,   8,  28,   8,   6],
       [  7,   3,  40,  72,  19,  20, 803,   6,  25,   5],
       [ 17,   4,  21,  24,  29,  40,   4, 839,   7,  15],
       [ 37,   5,   5,   4,   2,   2,   2,   3, 925,  15],
       [ 21,  70,   4,   8,   0,   1,   3,   8,  29, 856]])
```

```
# confusion matrix and accuracy
from sklearn.metrics import confusion_matrix, accuracy_score
plt.figure(figsize=(7, 6))
plt.title('Confusion matrix', fontsize=16)
plt.imshow(confusion_matrix(y_test, y_predictions1))
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```



```
[ ] # Define the class names for CIFAR-10
    class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

    # Choose a random image from the test set
    index = np.random.randint(0, x_test.shape[0])
    image = x_test[index]

    # Make a prediction on the image using the pre-trained model
    prediction = model.predict(np.expand_dims(image, axis=0))
    import tensorflow as tf

    class_index = tf.argmax(prediction, axis=1)
    class_name = tf.keras.utils.to_categorical(class_index, num_classes=10)
    actual_class = y_test[index]

    # Visualize the image and the predicted and actual classes
    plt.imshow(image)
    plt.title(f"Predicted class")
    plt.axis('off')
    plt.show()
```

```
1/1 [==============================] - 0s 28ms/step
```




```
[ ] from keras.models import load_model
    model = load_model('keras-VGG16-cifar10.h5')
    # predict
    plt.imshow(x_test[1000])

    result = model.predict(x_test[1000:1001]).tolist()
    predict = 0
    expect = y_test[1000][0]
    for i,_ in enumerate(result[0]):
        if result[0][i] > result[0][predict]:
            predict = i
```