Rishab Singh IS-3 2193192

In [1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
from tensorflow.keras import Sequential
from tensorflow.python.keras import regularizers
from tensorflow.keras.layers import Dense,Dropout,Activation,MaxPooling2D,Flatten,(
from tensorflow import keras
```

In [3]:
```
cifar10 = tf.keras.datasets.cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [4]:
```
print('X Training shape: ',X_train.shape)
print('Y Training shape: ',y_train.shape)
print('X Testing shape: ',X_test.shape)
print('Y Testing shape: ',y_test.shape)
```

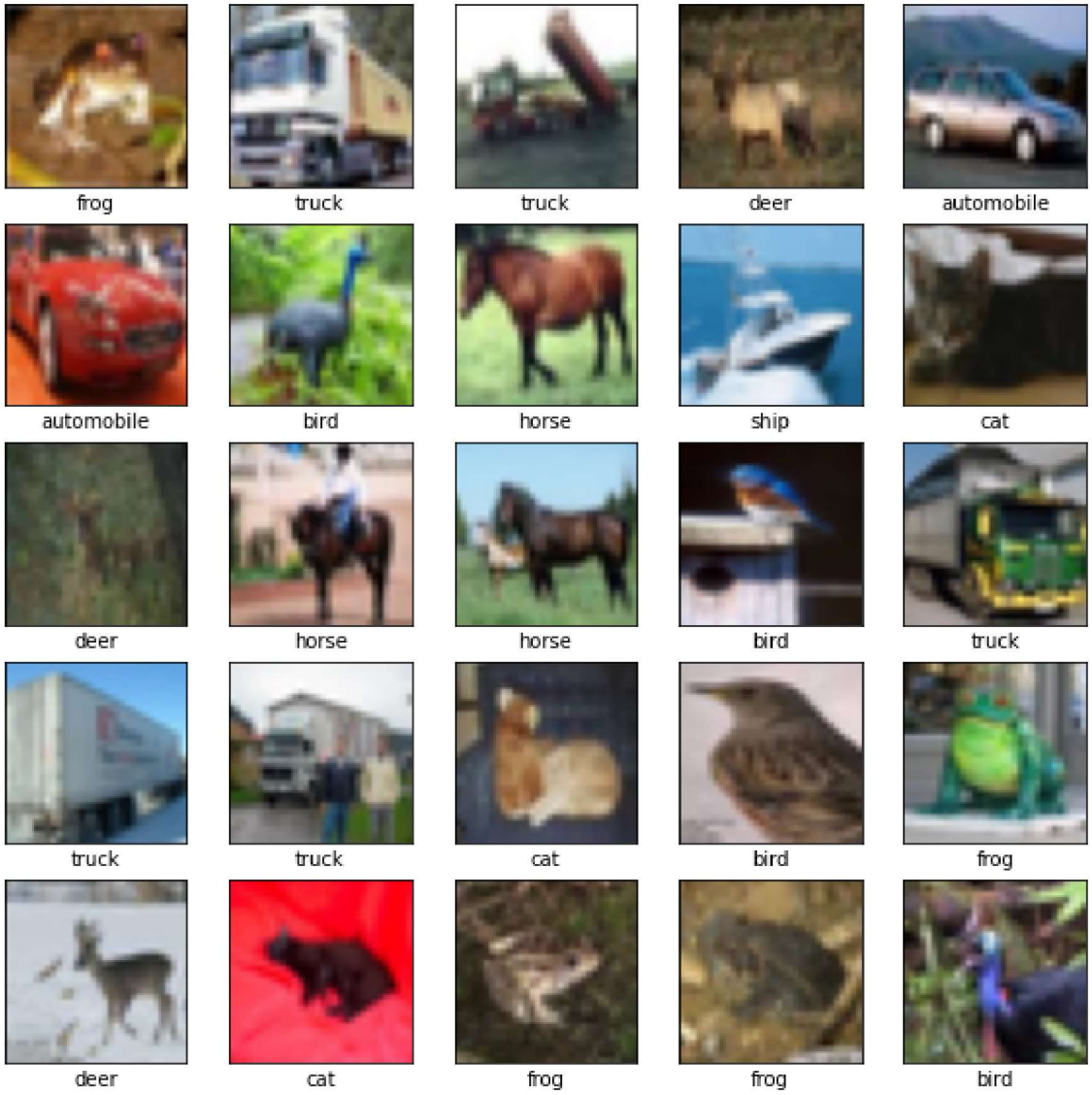
```
X Training shape:  (50000, 32, 32, 3)
Y Training shape:  (50000, 1)
X Testing shape:  (10000, 32, 32, 3)
Y Testing shape:  (10000, 1)
```

In [5]:
```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'hor
```

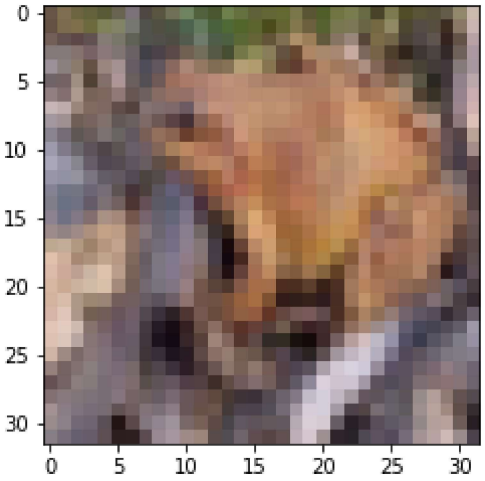
In [6]:
```
unique, counts = np.unique(y_train, return_counts=True)
print("Train labels: ", dict(zip(unique, counts)))


unique, counts = np.unique(y_test, return_counts=True)
print("\nTest labels: ", dict(zip(unique, counts)))
```

```
Train labels:  {0: 5000, 1: 5000, 2: 5000, 3: 5000, 4: 5000, 5: 5000, 6: 5000, 7:
5000, 8: 5000, 9: 5000}

Test labels:  {0: 1000, 1: 1000, 2: 1000, 3: 1000, 4: 1000, 5: 1000, 6: 1000, 7: 1
000, 8: 1000, 9: 1000}
```

In [7]:
```
# Visualizing some of the images from the training dataset
plt.figure(figsize=[10,10])
for i in range (25):
    # for first 25 images
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i][0]])

plt.show()
```



In [8]:
```
plt.imshow(X_train[204])
plt.show()
print("Digit in image is ",y_train[203])
```




```
Digit in image is  [3]
```

Data Preprocessing

In [9]:
```
X_train = X_train.reshape(50000, 32 * 32 * 3)
X_test = X_test.reshape(10000, 32 * 32 * 3)
X_train.shape
```

Out[9]:
```
(50000, 3072)
```

In [10]:
```
#normalising the data
print(X_train.max())
print(X_train.min())
X_train = X_train/255.0
X_test = X_test/255.0
print(X_train.max())
print(X_train.min())
```

```
255
0
1.0
0.0
```

In [11]:
```
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

In [12]:
```
X_train[200].shape
```

Out[12]:
```
(3072,)
```

In [13]:
```
model = Sequential()
model.add(Dense(1024,input_shape = (3072,),activation="relu"))
model.add(Dense(units = 512,kernel_regularizer=regularizers.l2(0.01),activation= "
model.add(Dense(units = 256,activation= "relu"))
model.add(Dropout(0.5))

model.add(Dense(units = 32,activation= "relu"))
model.add(Dense(units = 16,activation= "relu"))
model.add(Dense(units = 10,activation= "softmax"))
```

In [14]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1024)              3146752

 dense_1 (Dense)             (None, 512)               524800

 dense_2 (Dense)             (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_3 (Dense)             (None, 32)                8224

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 10)                170

=================================================================
Total params: 3,811,802
Trainable params: 3,811,802
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy
```

In [18]:
```
#training the model
```

```
history = model.fit(X_train, y_train, batch_size=128, epochs=20, validation_split=(
```

```
Epoch 1/20
313/313 [==============================] - 14s 46ms/step - loss: 2.0219 - accurac
y: 0.2506 - val_loss: 1.9180 - val_accuracy: 0.2933
Epoch 2/20
313/313 [==============================] - 15s 47ms/step - loss: 1.8829 - accurac
y: 0.3133 - val_loss: 1.8564 - val_accuracy: 0.3306
Epoch 3/20
313/313 [==============================] - 14s 44ms/step - loss: 1.8087 - accurac
y: 0.3491 - val_loss: 1.7944 - val_accuracy: 0.3500
Epoch 4/20
313/313 [==============================] - 14s 44ms/step - loss: 1.7376 - accurac
y: 0.3801 - val_loss: 1.7189 - val_accuracy: 0.3826
Epoch 5/20
313/313 [==============================] - 14s 44ms/step - loss: 1.6904 - accurac
y: 0.4000 - val_loss: 1.7433 - val_accuracy: 0.3822
Epoch 6/20
313/313 [==============================] - 14s 44ms/step - loss: 1.6464 - accurac
y: 0.4170 - val_loss: 1.6797 - val_accuracy: 0.4079
Epoch 7/20
313/313 [==============================] - 14s 44ms/step - loss: 1.6113 - accurac
y: 0.4311 - val_loss: 1.6283 - val_accuracy: 0.4270
Epoch 8/20
313/313 [==============================] - 14s 44ms/step - loss: 1.5901 - accurac
y: 0.4388 - val_loss: 1.6105 - val_accuracy: 0.4297
Epoch 9/20
313/313 [==============================] - 14s 44ms/step - loss: 1.5611 - accurac
y: 0.4476 - val_loss: 1.5646 - val_accuracy: 0.4463
Epoch 10/20
313/313 [==============================] - 14s 44ms/step - loss: 1.5525 - accurac
y: 0.4523 - val_loss: 1.5652 - val_accuracy: 0.4481
Epoch 11/20
313/313 [==============================] - 14s 45ms/step - loss: 1.5240 - accurac
y: 0.4633 - val_loss: 1.5518 - val_accuracy: 0.4531
Epoch 12/20
313/313 [==============================] - 16s 50ms/step - loss: 1.5135 - accurac
y: 0.4663 - val_loss: 1.5656 - val_accuracy: 0.4543
Epoch 13/20
313/313 [==============================] - 14s 45ms/step - loss: 1.4952 - accurac
y: 0.4743 - val_loss: 1.5408 - val_accuracy: 0.4585
Epoch 14/20
313/313 [==============================] - 14s 44ms/step - loss: 1.4754 - accurac
y: 0.4824 - val_loss: 1.5495 - val_accuracy: 0.4600
Epoch 15/20
313/313 [==============================] - 13s 43ms/step - loss: 1.4620 - accurac
y: 0.4881 - val_loss: 1.5270 - val_accuracy: 0.4665
Epoch 16/20
313/313 [==============================] - 13s 43ms/step - loss: 1.4555 - accurac
y: 0.4899 - val_loss: 1.5362 - val_accuracy: 0.4606
Epoch 17/20
313/313 [==============================] - 13s 43ms/step - loss: 1.4385 - accurac
y: 0.4962 - val_loss: 1.5127 - val_accuracy: 0.4723
Epoch 18/20
313/313 [==============================] - 14s 43ms/step - loss: 1.4296 - accurac
y: 0.5027 - val_loss: 1.5489 - val_accuracy: 0.4624
Epoch 19/20
313/313 [==============================] - 14s 43ms/step - loss: 1.4191 - accurac
y: 0.5030 - val_loss: 1.5082 - val_accuracy: 0.4713
Epoch 20/20
313/313 [==============================] - 14s 43ms/step - loss: 1.4022 - accurac
y: 0.5080 - val_loss: 1.4992 - val_accuracy: 0.4812
```

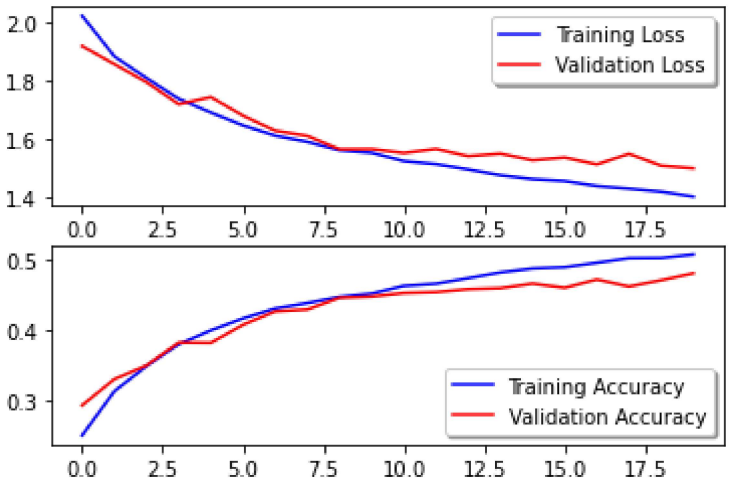
In [19]:
```
print("Training Accuracy")
```

```
print(model.evaluate(X_train,y_train))
print("Test accuracy")
print(model.evaluate(X_test,y_test))
```

```
Training Accuracy
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3808 - accurac
y: 0.5152
[1.380771517753601, 0.5151600241661072]
Test accuracy
313/313 [==============================] - 3s 9ms/step - loss: 1.4772 - accuracy:
0.4893
[1.4771984815597534, 0.489300012588501]
```

Plot accuracy and loss graph

In [20]:
```
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training Loss")
ax[0].plot(history.history['val_loss'], color='r', label="Validation Loss",axes =ax
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training Accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation Accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



Classification Report

In [24]:
```
from sklearn.metrics import classification_report
from sklearn import metrics
print(classification_report(Y_true, Y_pred_classes))
```

	precision	recall	f1-score	support
0	0.51	0.62	0.56	1000
1	0.56	0.67	0.61	1000
2	0.41	0.27	0.32	1000
3	0.35	0.27	0.30	1000
4	0.40	0.47	0.43	1000
5	0.46	0.35	0.40	1000
6	0.48	0.62	0.54	1000
7	0.60	0.50	0.54	1000
8	0.66	0.51	0.57	1000
9	0.46	0.64	0.53	1000
accuracy			0.49	10000
macro avg	0.49	0.49	0.48	10000
weighted avg	0.49	0.49	0.48	10000

In [ ]:

Save the Model

In [30]:
```
model.save("cifar_seq.h5")
```

In [ ]: