# Stoneburner, Kurt

- ## DSC 650 - Week 06
- ## Assignment 6.2 - ConvNet Model that classifies images in the CIFAR digital dataset.

This is mostly copy/pasted code from a website I found online. I've added notes to model layers to help with my understanding and includes notes on keras activation functions. I built the final model based on an evaluation of the validation model. I imported the plotting functions from a previous exercise.

Reference: https://keras.io/api/datasets/cifar10/ (https://keras.io/api/datasets/cifar10/)

CIFAR and Data augmentation https://stepup.ai/train_data_augmentation_keras/ (https://stepup.ai/train_data_augmentation_keras/)

Max-Pooling Explained: https://analyticsindiamag.com/max-pooling-in-convolutional-neural-network-and-its-features/ (https://analyticsindiamag.com/max-pooling-in-convolutional-neural-network-and-its-features/)

Conv2D Official Documentation: https://keras.io/api/layers/convolution_layers/convolution2d/ (https://keras.io/api/layers/convolution_layers/convolution2d/)

In [1]:

```
"""
#//*** Enable Plaid-Ml GPU backend for Radeon Cards. (it works faster
from os import environ

environ["KERAS_BACKEND"] = "plaidml.keras.backend"

import tensorflow.keras
"""
print()
```

In [2]:

```
import os
import sys
# //*** Imports and Load Data
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy

#//*** Use the whole window in the IPYNB editor
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

#//*** Maximize columns and rows displayed by pandas
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', None)
```

```python

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers,models

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
```

In [3]:

```python
#//*****************************************************
#//*** Plot a Fitted Models History of Loss and Accuracy
#//*****************************************************
def plot_model_history(input_history,loss='loss',acc='accuracy'):
    print(input_history.history.keys())
    loss = input_history.history[list(input_history.history.keys())[0
    acc = input_history.history[list(input_history.history.keys())[1]



    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, acc, "b", label="Training Accuracy")
    plt.title("Training Accuracy\nAccuracy should go up")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()


    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, "bo", label="Training Loss")

    plt.title("Training Loss \nLoss should go down")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

#//************************************************************
#//*** Plot a Fitted Models History Training and Validation Loss
#//************************************************************
def plot_model_validation(input_history):

    #//*** Assign Loss/Accuracy/Validation Loss/Validation Accuracy
    #//*** Based on Dictionary Key Tuple position

    loss = input_history.history[list(input_history.history.keys())[0
    acc = input_history.history[list(input_history.history.keys())[1]
    val_loss = input_history.history[list(input_history.history.keys(
    val_acc = input_history.history[list(input_history.history.keys()
    #print(loss,acc,val_loss,val_acc)
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, "bo", label="Training loss")
    plt.plot(epochs, val_loss, "b", label="Validation loss")
    plt.title("Training and validation loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
```

```
    plt.legend()
    plt.show()

    #//*** Plot the Validation Set Accuracy
    plt.clf()
    plt.plot(epochs, acc, "bo", label="Training accuracy")
    plt.plot(epochs, val_acc, "b", label="Validation accuracy")
    plt.title("Training and validation accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

def visualize_data(images, categories, class_names):
    fig = plt.figure(figsize=(14, 6))
    fig.patch.set_facecolor('white')
    for i in range(3 * 7):
        plt.subplot(3, 7, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(images[i])
        class_index = categories[i].argmax()
        plt.xlabel(class_names[class_index])
    plt.show()
```

### Assignment 6.2a

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. Do not use dropout or data-augmentation in this part.

Save the model, predictions, metrics, and validation plots in the dsc650/assignments /assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

In [4]:

```


#//*** Inputs reflects the shape of each individual piece of data.
#//*** The MNIST is a 28x28 single channel image.
#//*** The third is 1 channel. This is a greyscale image, therefore i
#//*** See Link above for further explanation

#//*** Conv2D: Filters defines the number of tensors at the layer. Ke
#//***          Conv2D reduces the image size by (filter_size - 1) in
#//*** MaxPooling2D: Is a form of feature reduction. In this case, Th
#//***               pool-size value (2x2) is kept. With a pool value
#//***               by 75%.
#//***               At each stage of max pooling, the image gets sma
#//***               for relationships between the reduced features.


```

Reference: [https://machinelearningknowledge.ai/keras-activation-layers-ultimate-guide-for-beginners/ (https://machinelearningknowledge.ai/keras-activation-layers-ultimate-guide-for-](https://machinelearningknowledge.ai/keras-activation-layers-ultimate-guide-for-beginners/)

beginners/)

## Keras Activation Cheat Sheet

**relu**: Rectified linear unit.

```
- ReLu activation function is computationally efficient hence it
enables neural networks to converge faster during the training ph
ase.
- It is both non-linear and differentiable which are good charact
eristics for activation function.
- ReLU does not suffer from the issue of Vanishing Gradient issue
like other activation functions and hence it is very effective in
hidden layers of large neural networks.

Data used in **ReLu** layers must be positive. Negative numbers c
annot be back-propagated and these values basically 'die'.
```

**softmax:** Generates a weighted value between 0 and 1. Use for multiple classification tasks.

**sigmoid:** Generates either 0 or 1. Data must be normalized. Does not handle outliers well Use for Binary Classification tasks.

**tanh:** Generates values from -1 to +1. Data must be normalized. Does not handle outliers well.

#### Which Activation Function to use in Neural Network?

**Sigmoid** and Tanh activation function should be avoided in hidden layers as they suffer from Vanishing Gradient problem.

**Sigmoid** activation function should be used in the output layer in case of Binary Classification

**ReLU** activation functions are ideal for hidden layers of neural networks as they do not suffer from the Vanishing Gradient problem and are computationally fast.

**Softmax** activation function should be used in the output layer in case of multiclass classification.

| Label | Description |
|---:|---:|
| 0 | airplane |
| 1 | automobile |
| 2 | bird |
| 3 | cat |
| 4 | deer |
| 5 | dog |
| 6 | frog |
| 7 | horse |
| 8 | ship |

**Label Description**

Returns

```
Tuple of NumPy arrays: (x_train, y_train), (x_test, y_test).
```

**x_train**: uint8 NumPy array of grayscale image data with shapes (50000, 32, 32, 3), containing the training data. Pixel values range from 0 to 255.

**y_train**: uint8 NumPy array of labels (integers in range 0-9) with shape (50000, 1) for the training data.

**x_test**: uint8 NumPy array of grayscale image data with shapes (10000, 32, 32, 3), containing the test data. Pixel values range from 0 to 255.

**y_test**: uint8 NumPy array of labels (integers in range 0-9) with shape (10000, 1) for the test data.

In [5]:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog'
num_classes = len(class_names)

#//*** Load the CIFAR10 dataset into the default test train splits
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_


#//*** Build a subset of the data for model exploration
subset_size = 5000

#//*** Generate Partial validation sets
x_validation = x_train[subset_size:subset_size*2]
y_validation = y_train[subset_size:subset_size*2]

#//*** Validation subset
x_train = x_train[:subset_size]
y_train = y_train[:subset_size]


print(x_train.shape)
print(y_train.shape)

#//*** Verify the subsets are the proper shape
assert x_train.shape == (subset_size, 32, 32, 3)
assert y_train.shape == (subset_size, 1)

#//*** Max test size is 10000
if subset_size > 10000:
    subset_size = 10000

x_test = x_test[:subset_size]
y_test = y_test[:subset_size]

print(x_test.shape)
print(y_test.shape)

#//*** Verify the subsets are the proper shape
```

```
assert x_test.shape == (subset_size, 32, 32, 3)
assert y_test.shape == (subset_size, 1)


x_train = x_train / 255.0
y_train = to_categorical(y_train, num_classes)

x_test = x_test / 255.0
y_test = to_categorical(y_test, num_classes)

visualize_data(x_train, y_train, class_names)
```

```
(5000, 32, 32, 3)
(5000, 1)
(5000, 32, 32, 3)
(5000, 1)
```


In [6]:

```
def create_model():

    #//*** Using relu (Rectified Linear Units) for the hidden layers.
    #//*** Using Convolution2d kernal_size =3, create unique features
    #//*** MaxPooling 2, takes the 3x3 feature pixels (which are now
    #//*** This reduces the features by 75% per feature group. This i

    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='s
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='s
    model.add(layers.MaxPool2D((2,2)))

    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='s
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='s
    model.add(layers.MaxPool2D((2,2)))

    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='
    model.add(layers.MaxPool2D((2,2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))

```

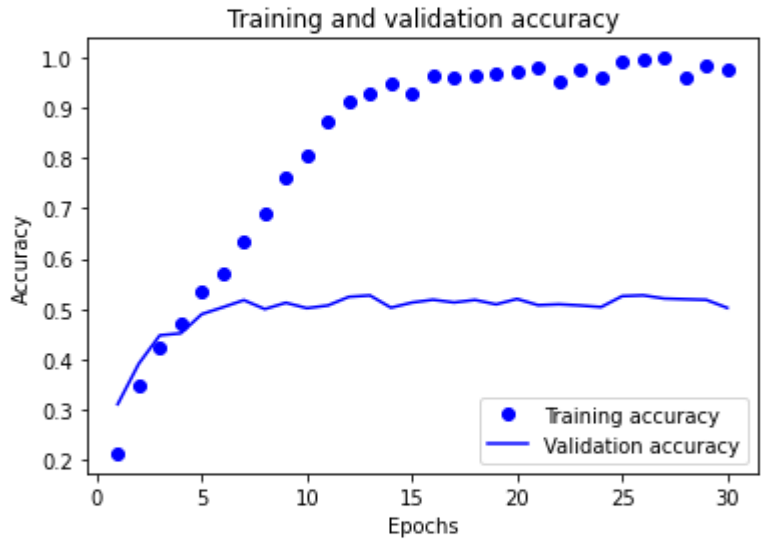
```
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

In [7]:

```
batch_size = 32
epochs = 30
m_no_aug = create_model()
m_no_aug.summary()

history_no_aug = m_no_aug.fit(
    x_train, y_train,
    epochs=epochs, batch_size=batch_size,
    validation_data=(x_test, y_test))

loss_no_aug, acc_no_aug = m_no_aug.evaluate(x_test,  y_test)

print(f"Test accuracy: {acc_no_aug:.3f}")



plot_model_validation(history_no_aug)
```

```
WARNING:tensorflow:From C:\Users\family\anaconda3\envs\directml\lib\si
te-packages\tensorflow_core\python\ops\resource_variable_ops.py:1630:
calling BaseResourceVariable.__init__ (from tensorflow.python.ops.reso
urce_variable_ops) with constraint is deprecated and will be removed i
n a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
Train on 5000 samples, validate on 5000 samples
Epoch 1/30
5000/5000 [==============================] - 4s 869us/sample - loss:
2.0975 - acc: 0.2126 - val_loss: 1.8855 - val_acc: 0.3104
Epoch 2/30
5000/5000 [==============================] - 3s 657us/sample - loss:
1.7665 - acc: 0.3472 - val_loss: 1.7073 - val_acc: 0.3908
Epoch 3/30
5000/5000 [==============================] - 3s 656us/sample - loss:
1.5784 - acc: 0.4224 - val_loss: 1.5296 - val_acc: 0.4474
Epoch 4/30
5000/5000 [==============================] - 3s 660us/sample - loss:
1.4523 - acc: 0.4720 - val_loss: 1.5033 - val_acc: 0.4514
Epoch 5/30
5000/5000 [==============================] - 3s 660us/sample - loss:
1.2895 - acc: 0.5338 - val_loss: 1.4685 - val_acc: 0.4896
Epoch 6/30
```

```
5000/5000 [==============================] - 3s 661us/sample - loss:
1.1680 - acc: 0.5708 - val_loss: 1.3664 - val_acc: 0.5032
Epoch 7/30
5000/5000 [==============================] - 3s 656us/sample - loss:
1.0337 - acc: 0.6328 - val_loss: 1.4326 - val_acc: 0.5174
Epoch 8/30
5000/5000 [==============================] - 3s 654us/sample - loss:
0.8545 - acc: 0.6914 - val_loss: 1.5443 - val_acc: 0.4996
Epoch 9/30
5000/5000 [==============================] - 3s 653us/sample - loss:
0.6645 - acc: 0.7600 - val_loss: 1.7166 - val_acc: 0.5122
Epoch 10/30
5000/5000 [==============================] - 3s 656us/sample - loss:
0.5287 - acc: 0.8070 - val_loss: 1.7572 - val_acc: 0.5014
Epoch 11/30
5000/5000 [==============================] - 3s 657us/sample - loss:
0.3674 - acc: 0.8742 - val_loss: 1.9354 - val_acc: 0.5068
Epoch 12/30
5000/5000 [==============================] - 3s 653us/sample - loss:
0.2485 - acc: 0.9124 - val_loss: 2.2690 - val_acc: 0.5240
Epoch 13/30
5000/5000 [==============================] - 3s 656us/sample - loss:
0.2210 - acc: 0.9268 - val_loss: 2.8940 - val_acc: 0.5268
Epoch 14/30
5000/5000 [==============================] - 3s 660us/sample - loss:
0.1439 - acc: 0.9504 - val_loss: 3.0321 - val_acc: 0.5024
Epoch 15/30
5000/5000 [==============================] - 3s 658us/sample - loss:
0.2157 - acc: 0.9272 - val_loss: 2.7528 - val_acc: 0.5126
Epoch 16/30
5000/5000 [==============================] - 3s 661us/sample - loss:
0.1037 - acc: 0.9638 - val_loss: 3.0390 - val_acc: 0.5182
Epoch 17/30
5000/5000 [==============================] - 3s 658us/sample - loss:
0.1160 - acc: 0.9610 - val_loss: 3.5346 - val_acc: 0.5128
Epoch 18/30
5000/5000 [==============================] - 3s 654us/sample - loss:
0.1069 - acc: 0.9632 - val_loss: 3.4819 - val_acc: 0.5178
Epoch 19/30
5000/5000 [==============================] - 3s 658us/sample - loss:
0.0992 - acc: 0.9670 - val_loss: 3.1979 - val_acc: 0.5090
Epoch 20/30
5000/5000 [==============================] - 3s 656us/sample - loss:
```




# Run No-Augmentation Model with full dataset

14 Epochs is seems to be a good balance between fitting and over fitting - Using 5000 Samples.
22 Epochs was a good value when testing with 1000 Samples.

In [8]:

```
#//*** Reload full data set
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog'
num_classes = len(class_names)

#//*** Load the CIFAR10 dataset into the default test train splits
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_


#//*** Verify the subsets are the proper shape
assert x_train.shape == (50000, 32, 32, 3)
assert y_train.shape == (50000, 1)

#//*** Verify the subsets are the proper shape
assert x_test.shape == (10000, 32, 32, 3)
assert y_test.shape == (10000, 1)

#//*** Not sure why x_train is divided by 255 or why Y_train is conve
x_train = x_train / 255.0
y_train = to_categorical(y_train, num_classes)

x_test = x_test / 255.0
y_test = to_categorical(y_test, num_classes)

#//*** Display Sample of Data
visualize_data(x_train, y_train, class_names)
```

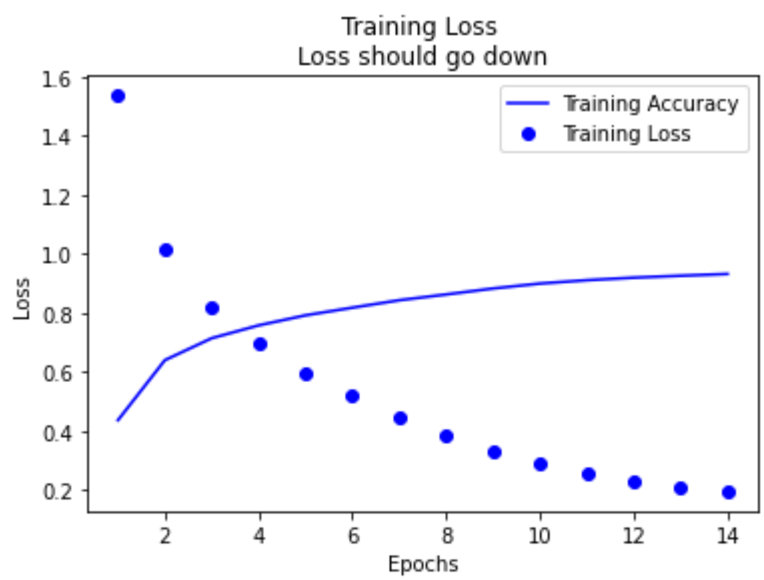

In [9]:

```
#//*** 14 Epochs to prevent overfitting
batch_size = 32
epochs = 14
m_no_aug = create_model()
m_no_aug.summary()

history_no_aug = m_no_aug.fit(
    x_train, y_train,
    epochs=epochs, batch_size=batch_size,
    validation_data=(x_test, y_test))

loss_no_aug, acc_no_aug = m_no_aug.evaluate(x_test,  y_test)

print(f"Model accuracy: {acc_no_aug:.3f}")
```

```
plot_model_history(history_no_aug)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 128)         73856     
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               262272    
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
Train on 50000 samples, validate on 10000 samples
Epoch 1/14
50000/50000 [==============================] - 26s 510us/sample - los
s: 1.5360 - acc: 0.4358 - val_loss: 1.1374 - val_acc: 0.5962
Epoch 2/14
50000/50000 [==============================] - 25s 506us/sample - los
s: 1.0162 - acc: 0.6391 - val_loss: 0.9327 - val_acc: 0.6672
Epoch 3/14
50000/50000 [==============================] - 25s 508us/sample - los
s: 0.8172 - acc: 0.7136 - val_loss: 0.8484 - val_acc: 0.7025
Epoch 4/14
50000/50000 [==============================] - 26s 512us/sample - los
s: 0.6937 - acc: 0.7570 - val_loss: 0.8370 - val_acc: 0.7165
Epoch 5/14
50000/50000 [==============================] - 26s 511us/sample - los
s: 0.5950 - acc: 0.7911 - val_loss: 0.8198 - val_acc: 0.7286
Epoch 6/14
50000/50000 [==============================] - 25s 507us/sample - los
s: 0.5169 - acc: 0.8170 - val_loss: 0.7811 - val_acc: 0.7398
Epoch 7/14
50000/50000 [==============================] - 25s 510us/sample - los
```

```
s: 0.4428 - acc: 0.8422 - val_loss: 0.8098 - val_acc: 0.7388
Epoch 8/14
50000/50000 [==============================] - 25s 508us/sample - los
s: 0.3873 - acc: 0.8618 - val_loss: 0.8176 - val_acc: 0.7438
Epoch 9/14
50000/50000 [==============================] - 25s 506us/sample - los
s: 0.3289 - acc: 0.8820 - val_loss: 0.8895 - val_acc: 0.7450
Epoch 10/14
50000/50000 [==============================] - 25s 506us/sample - los
s: 0.2867 - acc: 0.8988 - val_loss: 0.9468 - val_acc: 0.7487
Epoch 11/14
50000/50000 [==============================] - 25s 507us/sample - los
s: 0.2551 - acc: 0.9101 - val_loss: 1.0424 - val_acc: 0.7423
Epoch 12/14
50000/50000 [==============================] - 25s 506us/sample - los
s: 0.2286 - acc: 0.9188 - val_loss: 1.1135 - val_acc: 0.7479
Epoch 13/14
50000/50000 [==============================] - 25s 506us/sample - los
s: 0.2111 - acc: 0.9253 - val_loss: 1.1478 - val_acc: 0.7442
Epoch 14/14
50000/50000 [==============================] - 25s 506us/sample - los
s: 0.1947 - acc: 0.9312 - val_loss: 1.1876 - val_acc: 0.7382
10000/10000 [==============================] - 2s 191us/sample - loss:
1.1876 - acc: 0.7382
Model accuracy: 0.738
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```




# Assignment 6.2.b

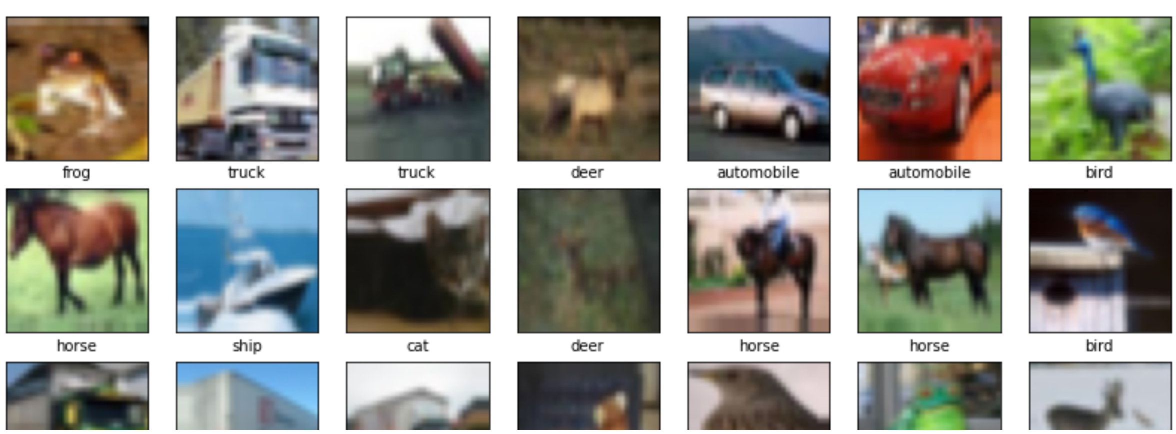
Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. This time includes dropout and data-augmentation.

In [10]:

```
import scipy
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog'
num_classes = len(class_names)
```

```

#//*** Load the CIFAR10 dataset into the default test train splits
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load

x_train = x_train / 255.0
y_train = to_categorical(y_train, num_classes)

x_test = x_test / 255.0
y_test = to_categorical(y_test, num_classes)

visualize_data(x_train, y_train, class_names)

width_shift = 3/32
height_shift = 3/32
flip = True

datagen = ImageDataGenerator(
    horizontal_flip=flip,
    width_shift_range=width_shift,
    height_shift_range=height_shift,
    )
datagen.fit(x_train)

it = datagen.flow(x_train, y_train, shuffle=False)
batch_images, batch_labels = next(it)
print("Transformed images")
visualize_data(batch_images, batch_labels, class_names)
```


Transformed images



In [11]:

```
batch_size = 32
epochs = 14
model = create_model()
model.summary()

history = model.fit(
    x_train, y_train,
    epochs=epochs, batch_size=batch_size,
    #validation_data=(x_test, y_test)
)

loss, acc = model.evaluate(batch_images,  batch_labels)

print(f"Model accuracy: {acc:.3f}")


plot_model_history(history)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
Train on 50000 samples
Epoch 1/14
50000/50000 [==============================] - 24s 470us/sample - los
s: 1.4707 - acc: 0.4629
Epoch 2/14
50000/50000 [==============================] - 23s 467us/sample - los
s: 0.9800 - acc: 0.6520
Epoch 3/14
50000/50000 [==============================] - 23s 468us/sample - los
```

In [ ]:

In [ ]:

In [ ]: