ECE 176 Assignment 2: Linear Regression

For this part of assignment, you are tasked to implement a linear regression algorithm for multiclass classification and test it on the CIFAR10 dataset.

You sould run the whole notebook and answer the questions in the notebook.

CIFAR 10 dataset contains 32x32x3 RGB images of 10 distinct cateogaries, and our aim is to predict which class the image belongs to

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

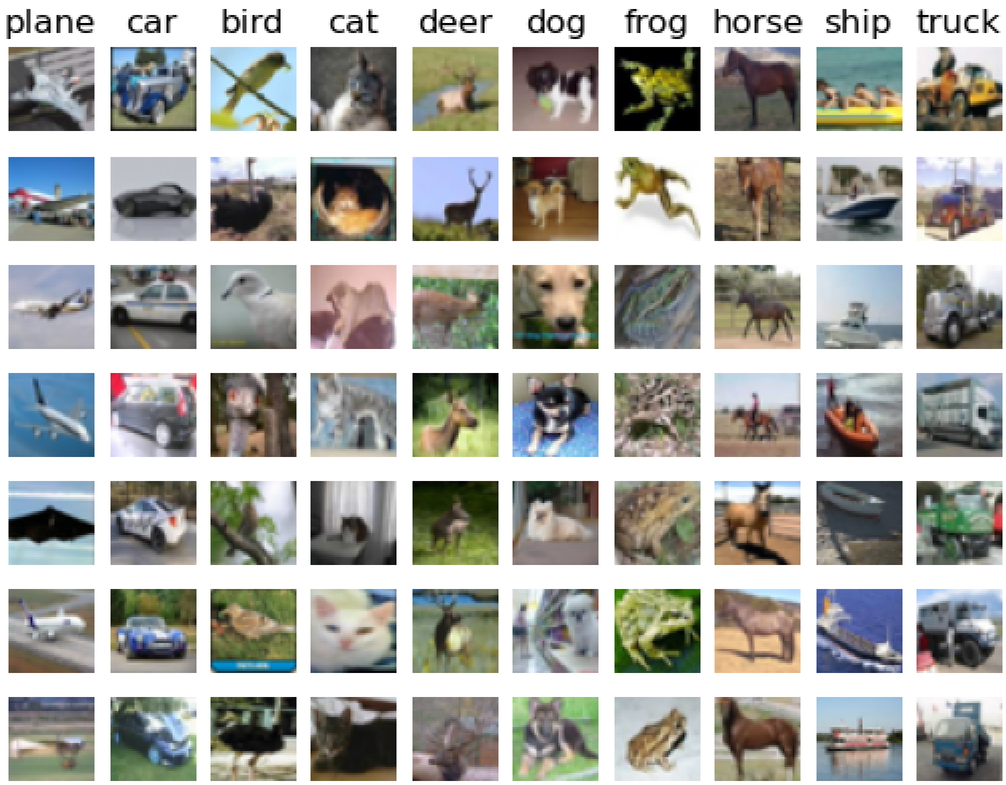
In [2]:
```
# Prepare Packages
import numpy as np
import matplotlib.pyplot as plt

from utils.data_processing import get_cifar10_data

# Use a subset of CIFAR10 for the assignment
dataset = get_cifar10_data(
    subset_train=5000,
    subset_val=250,
    subset_test=500,
)

print(dataset.keys())
print("Training Set Data  Shape: ", dataset["x_train"].shape)
print("Training Set Label Shape: ", dataset["y_train"].shape)
print("Validation Set Data  Shape: ", dataset["x_val"].shape)
print("Validation Set Label Shape: ", dataset["y_val"].shape)
print("Test Set Data  Shape: ", dataset["x_test"].shape)
print("Test Set Label Shape: ", dataset["y_test"].shape)
```

```
dict_keys(['x_train', 'y_train', 'x_val', 'y_val', 'x_test', 'y_test'])
Training Set Data  Shape:  (5000, 3072)
Training Set Label Shape:  (5000,)
Validation Set Data  Shape:  (250, 3072)
Validation Set Label Shape:  (250,)
Test Set Data  Shape:  (500, 3072)
Test Set Label Shape:  (500,)
```

In [4]:
```
x_train = dataset["x_train"]
y_train = dataset["y_train"]
x_val = dataset["x_val"]
y_val = dataset["y_val"]
x_test = dataset["x_test"]
y_test = dataset["y_test"]
```

In [6]:
```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = [
```

```
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
samples_per_class = 7


def visualize_data(dataset, classes, samples_per_class):
    num_classes = len(classes)
    for y, cls in enumerate(classes):
        idxs = np.flatnonzero(y_train == y)
        idxs = np.random.choice(idxs, samples_per_class, replace=False)
        for i, idx in enumerate(idxs):
            plt_idx = i * num_classes + y + 1
            plt.subplot(samples_per_class, num_classes, plt_idx)
            plt.imshow(dataset[idx])
            plt.axis("off")
            if i == 0:
                plt.title(cls)
    plt.show()


visualize_data(
    x_train.reshape(5000, 3, 32, 32).transpose(0, 2, 3, 1), classes, samples
)
```



Linear Regression for multi-class classification

A Linear Regression Algorithm has these hyperparameters:

- **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, and later you are asked to experiment with different values. We recommend looking at the graphs and observing how the performance of the classifier changes with different learning rate.
- **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according the linear classifier update rule for each sample in the training set. We evaluate our models after every 10 epochs and save the accuracies, which are later used to plot the training, validation and test VS epoch curves.
- **Weight Decay** - Regularization can be used to constrain the weights of the classifier and prevent their values from blowing up. Regularization helps in combatting overfitting. You will be using the 'weight_decay' term to introduce regularization in the classifier.

Implementation (50%)

You first need to implement the Linear Regression method in `algorithms/linear_regression.py`. The formulations follow the lecture (consider binary classification for each of the 10 classes, with labels -1 / 1 for not belonging / belonging to the class). You need to fill in the training function as well as the prediction function.

In [11]:
```
# Import the algorithm implementation (TODO: Complete the Linear Regression
from algorithms import Linear
from utils.evaluation import get_classification_accuracy

num_classes = 10  # Cifar10 dataset has 10 different classes

# Initialize hyper-parameters
learning_rate = 0.0001  # You will be later asked to experiment with differe
num_epochs_total = 200  # Total number of epochs to train the classifier
epochs_per_evaluation = 10  # Epochs per step of evaluation; We will evaluat
N, D = dataset[
    "x_train"
].shape  # Get training data shape, N: Number of examples, D:Dimensionality
weight_decay = 0.0

# Insert additional scalar term 1 in the samples to account for the bias as
x_train = np.insert(x_train, D, values=1, axis=1)
x_val = np.insert(x_val, D, values=1, axis=1)
x_test = np.insert(x_test, D, values=1, axis=1)
```

In [13]:
```
# Training and evaluation function -> Outputs accuracy data
def train(learning_rate_, weight_decay_):
    # Create a linear regression object
    linear_regression = Linear(
        num_classes, learning_rate_, epochs_per_evaluation, weight_decay_
    )

    # Randomly initialize the weights and biases
    weights = np.random.randn(num_classes, D + 1) * 0.0001

    train_accuracies, val_accuracies, test_accuracies = [], [], []

    # Train the classifier
    for _ in range(int(num_epochs_total / epochs_per_evaluation)):
        # Train the classifier on the training data
        weights = linear_regression.train(x_train, y_train, weights)

        # Evaluate the trained classifier on the training dataset
        y_pred_train = linear_regression.predict(x_train)
        train_accuracies.append(get_classification_accuracy(y_pred_train, y_

        # Evaluate the trained classifier on the validation dataset
        y_pred_val = linear_regression.predict(x_val)
        val_accuracies.append(get_classification_accuracy(y_pred_val, y_val)

        # Evaluate the trained classifier on the test dataset
        y_pred_test = linear_regression.predict(x_test)
        test_accuracies.append(get_classification_accuracy(y_pred_test, y_te
```

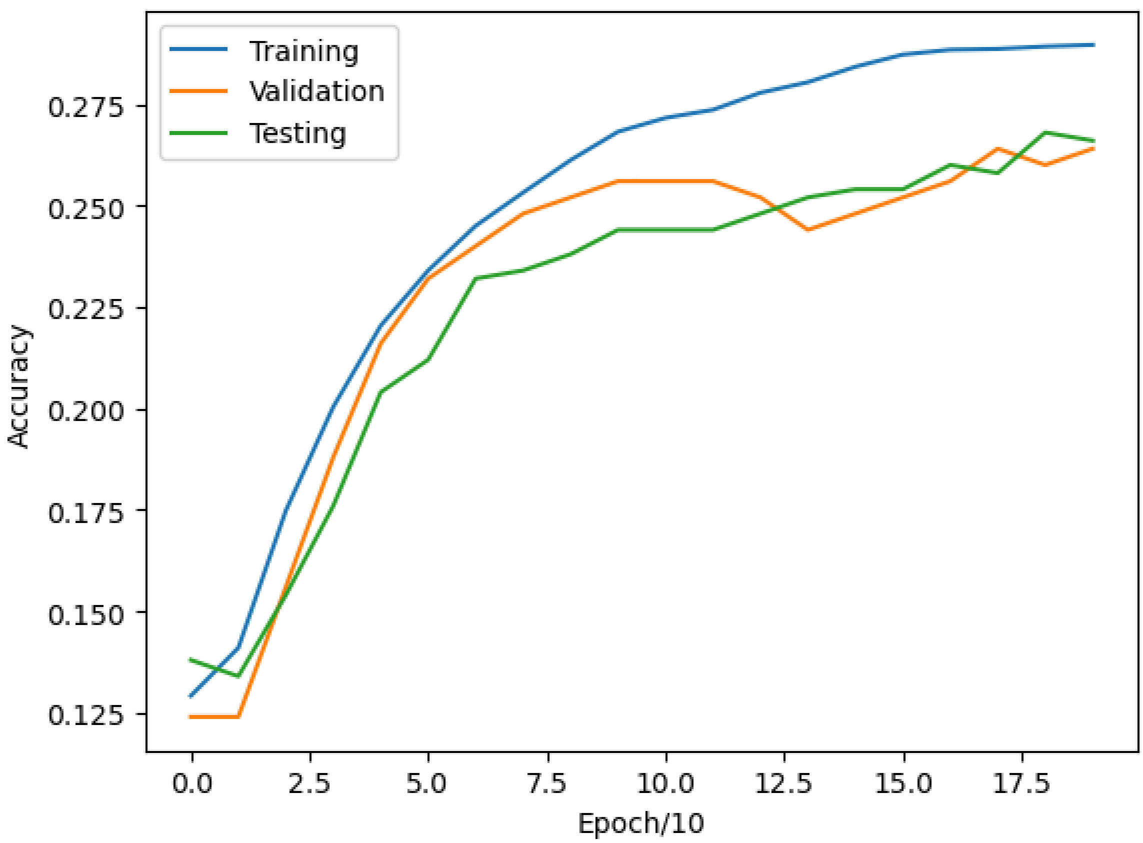
```
    return train_accuracies, val_accuracies, test_accuracies, weights
```

Plot the Accuracies vs epoch graphs

In [16]:
```
import matplotlib.pyplot as plt


def plot_accuracies(train_acc, val_acc, test_acc):
    # Plot Accuracies vs Epochs graph for all the three
    epochs = np.arange(0, int(num_epochs_total / epochs_per_evaluation))
    plt.ylabel("Accuracy")
    plt.xlabel("Epoch/10")
    plt.plot(epochs, train_acc, epochs, val_acc, epochs, test_acc)
    plt.legend(["Training", "Validation", "Testing"])
    plt.show()
```

In [18]:
```
# Run training and plotting for default parameter values as mentioned above
t_ac, v_ac, te_ac, weights = train(learning_rate, weight_decay)
```

In [19]:
```
plot_accuracies(t_ac, v_ac, te_ac)
```

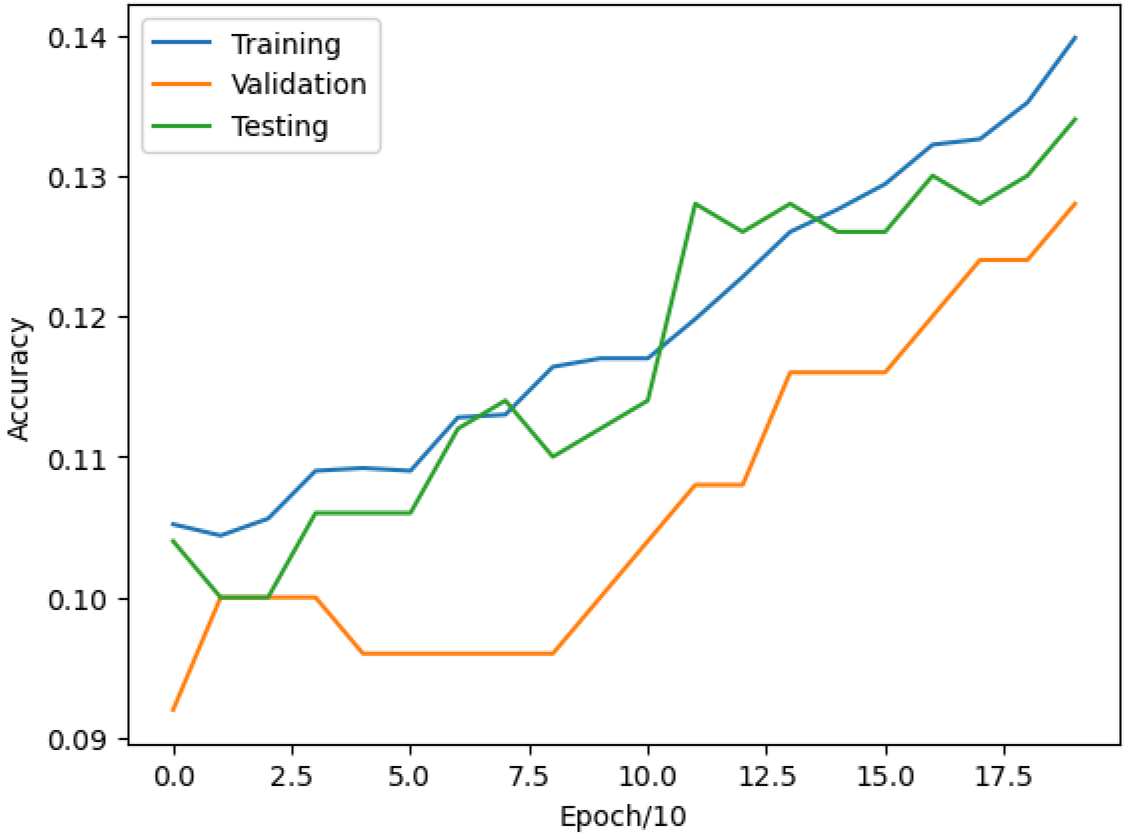


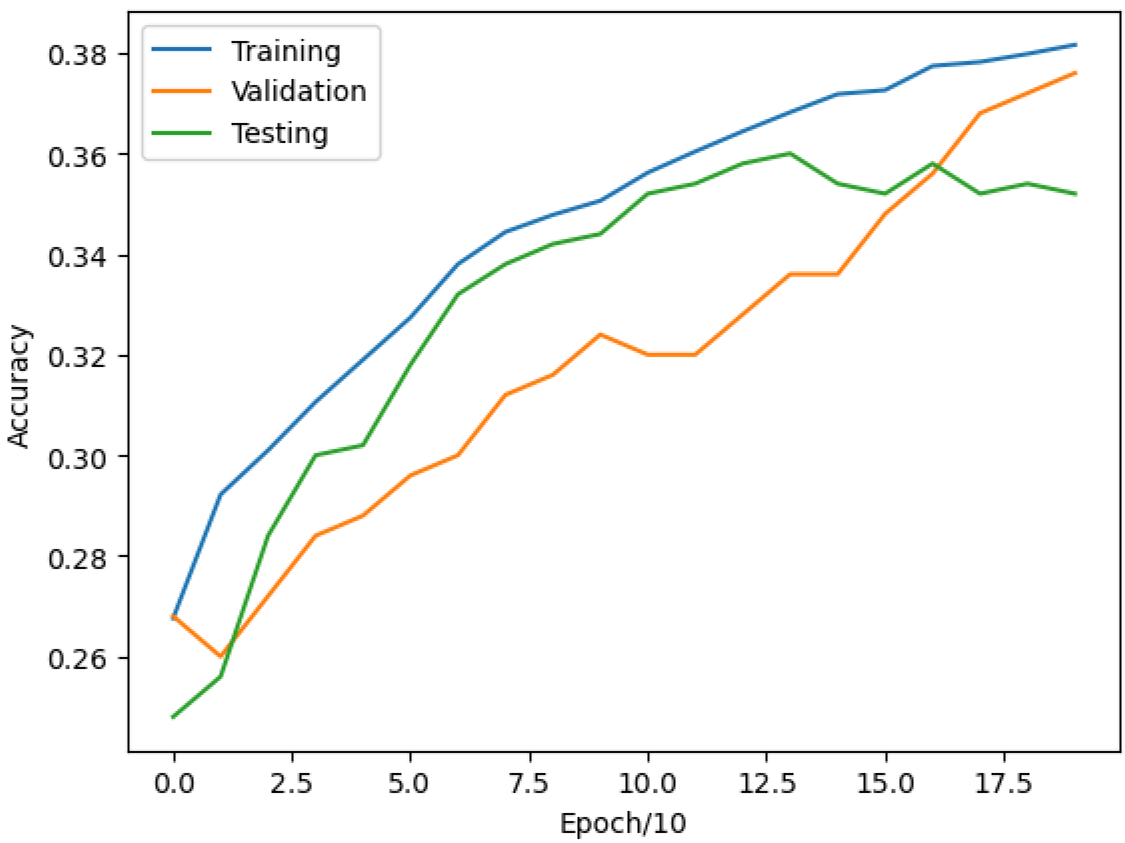
Try different learning rates and plot graphs for all (20%)

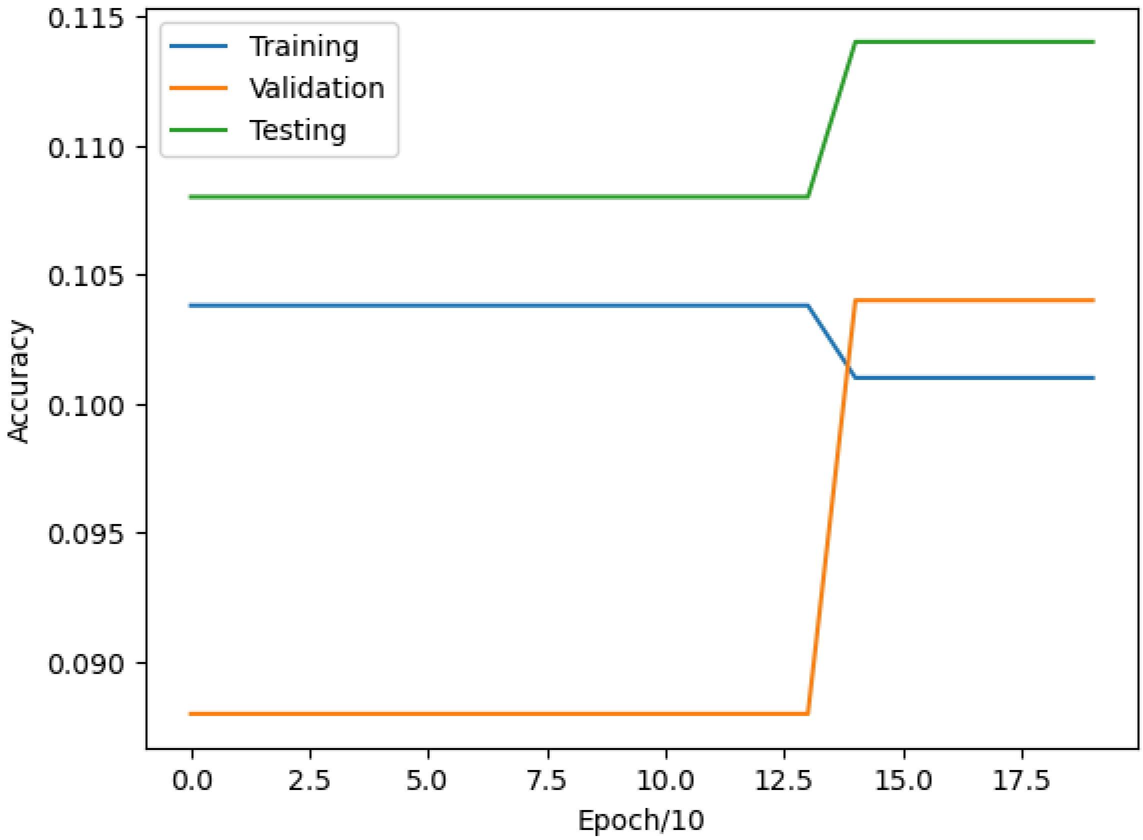
In [57]:
```
# Initialize the best values
best_weights = weights
best_learning_rate = learning_rate
```

```
best_weight_decay = weight_decay

# TODO
# Repeat the above training and evaluation steps for the following learning 
# You need to try 3 learning rates and submit all 3 graphs along with this r
learning_rates = [0.00001,0.001,0.1]
weight_decay = 0.0  # No regularization for now

# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF THEY ACH

# for lr in learning_rates: Train the classifier and plot data
# Step 1. train_accu, val_accu, test_accu = train(lr, weight_decay)
# Step 2. plot_accuracies(train_accu, val_accu, test_accu)
train_accuracies=[]
val_accuracies=[]
test_accuracies=[]
for learning_rate in learning_rates:
    train_accu, val_accu, test_accu, weights = train(learning_rate, weight_d

    train_accuracies.append(train_accu)
    val_accuracies.append(val_accu)
    test_accuracies.append(test_accu)

    print(f"Learning rate:{learning_rate:.6f}")

plot_accuracies(train_accuracies[0], val_accuracies[0], test_accuracies[0])
plot_accuracies(train_accuracies[1], val_accuracies[1], test_accuracies[1])
plot_accuracies(train_accuracies[2], val_accuracies[2], test_accuracies[2])


    # TODO: Train the classifier with different learning rates and plot
    #pass
```

```
Learning rate:0.000010
Learning rate:0.001000
Learning rate:0.100000
```


Training
Validation
Testing
Accuracy
Epoch/10
0.09
0.10
0.11
0.12
0.13
0.14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training
Validation
Testing
Accuracy
Epoch/10
0.26
0.28
0.30
0.32
0.34
0.36
0.38
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Inline Question 1.

Which one of these learning rates (best_lr) would you pick to train your model? Please Explain why.

Your Answer:

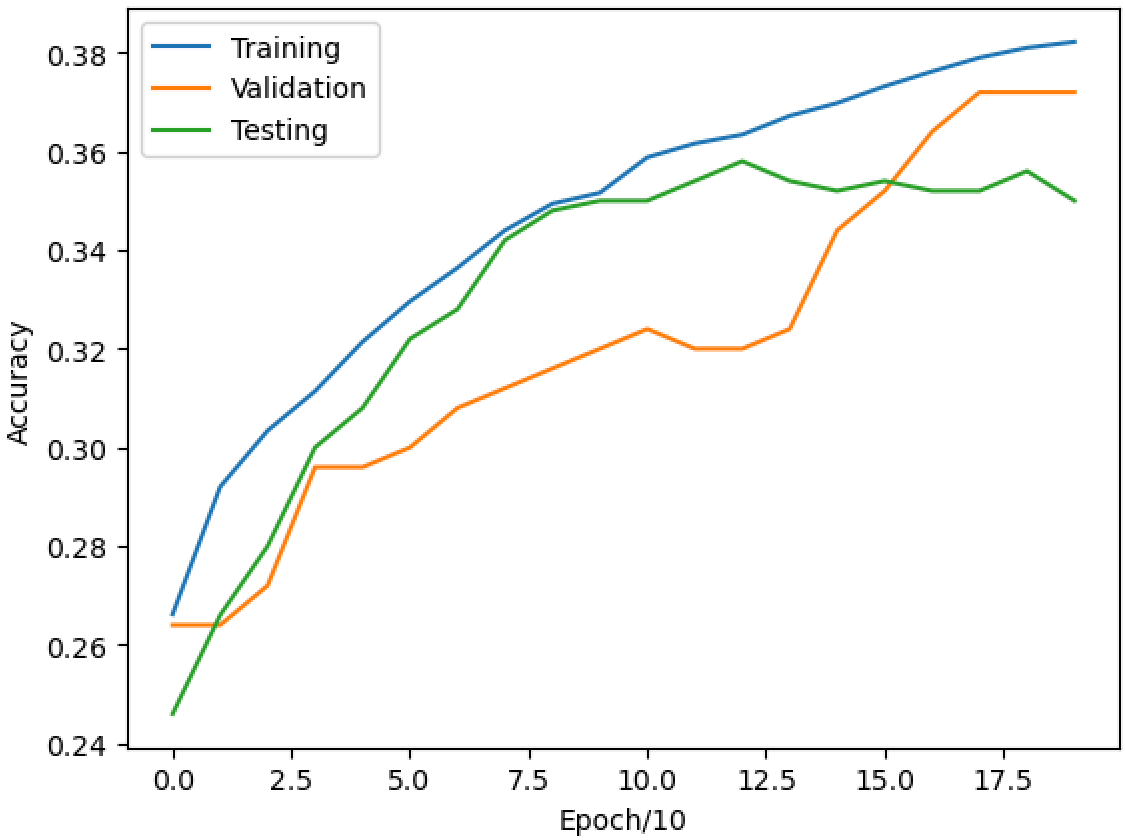
Learning rate=0.001(second graph) is the best choice among the three rates(0.00001,0.001,0.01) tested. It achieved the highest accuracy of around 0.36~0.38 and show steady,smooth improvement in both training and validation curves. While 0.00001 learns too slow and plateaus at a lower accuracy of 0.13, 0.01 is too high and the results in almost no learning with accuracy stuck around 0.1-0.115. The middle rate 0.001 provides the optimal balance between learning speed, stability, and final performance, making it the most effective choice for training the model.

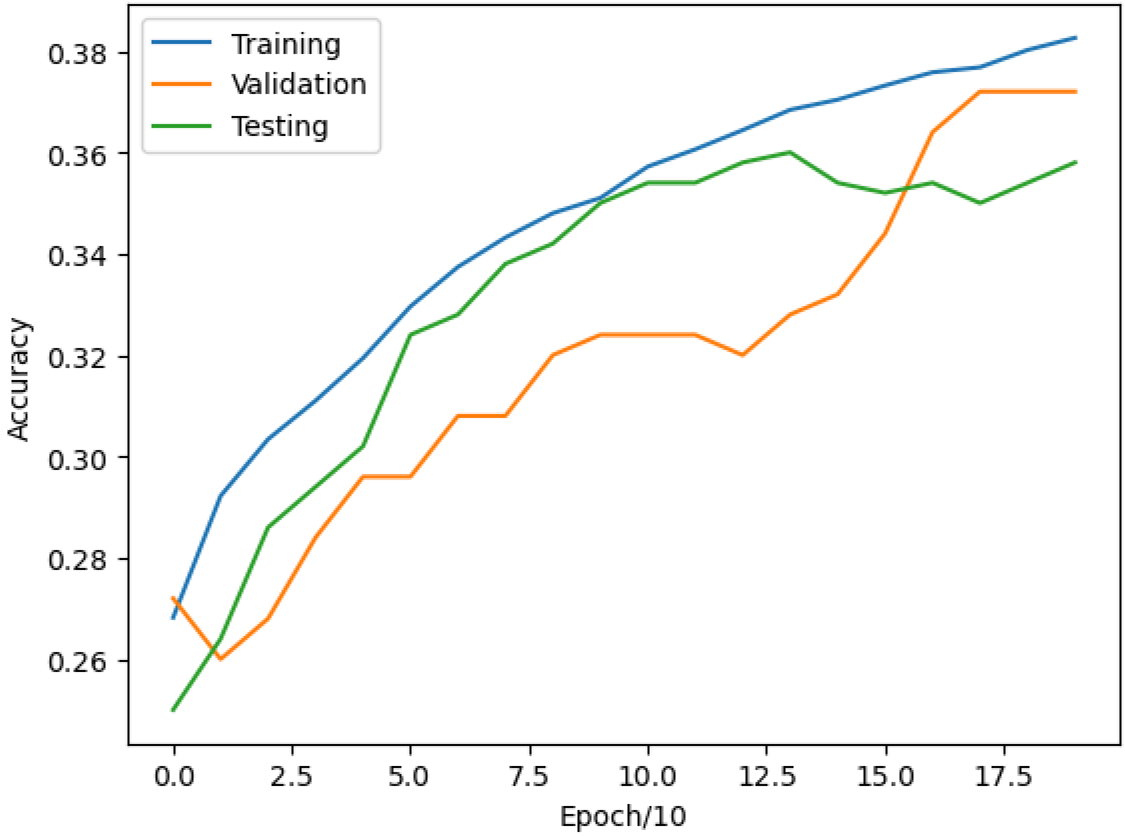
Regularization: Try different weight decay and plot graphs for all (20%)

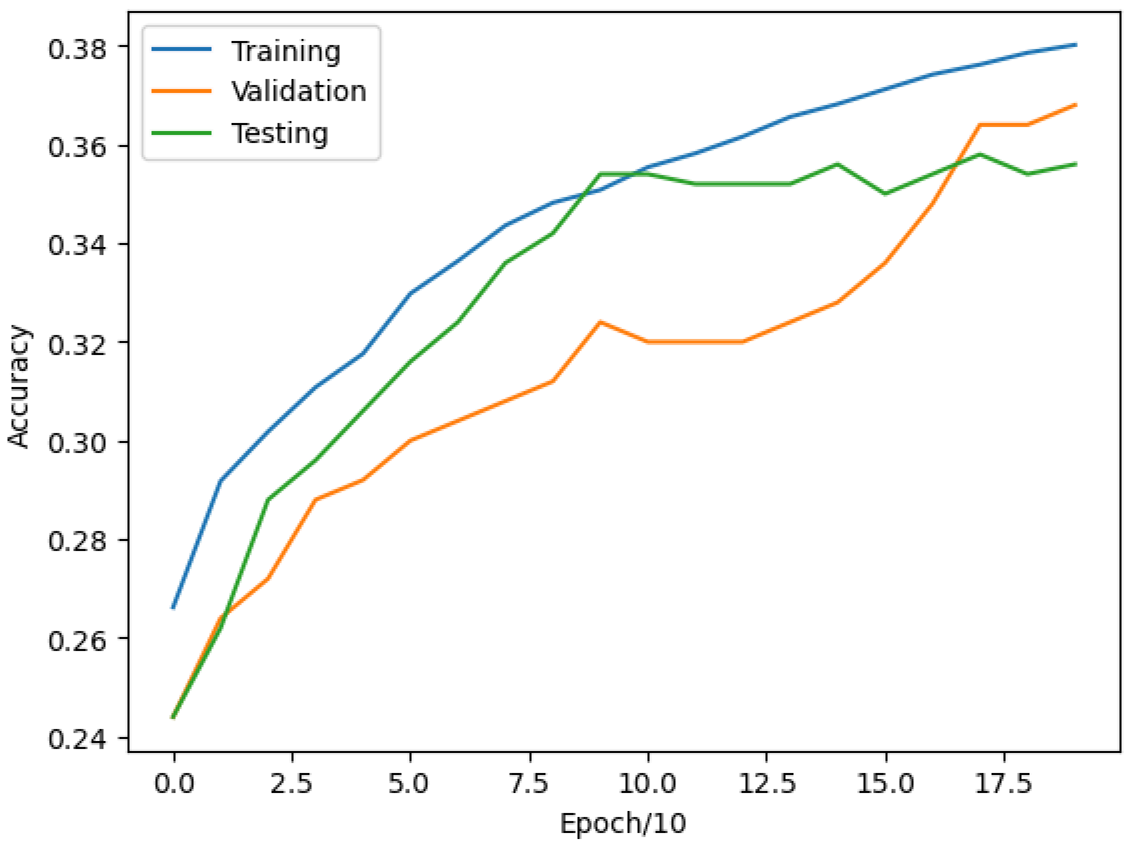
In [68]:
```
# Initialize a non-zero weight_decay (Regularization constant) term and repe
# Use the best learning rate as obtained from the above exercise, best_lr

# You need to try 3 learning rates and submit all 3 graphs along with this r
weight_decays = [0.00001,0.001,1]
learning_rate=0.001
# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF THEY ACH
```

```
# for weight_decay in weight_decays: Train the classifier and plot data
# Step 1. train_accu, val_accu, test_accu = train(best_lr, weight_decay)
# Step 2. plot_accuracies(train_accu, val_accu, test_accu)
train_accuracies=[]
val_accuracies=[]
test_accuracies=[]
for weight_decay in weight_decays:
    train_accu, val_accu, test_accu, weights = train(learning_rate, weight_d
    train_accuracies.append(train_accu)
    val_accuracies.append(val_accu)
    test_accuracies.append(test_accu)

    print(f"weigh_decay:{weight_decay:.6f}")
plot_accuracies(train_accuracies[0], val_accuracies[0], test_accuracies[0])
plot_accuracies(train_accuracies[1], val_accuracies[1], test_accuracies[1])
plot_accuracies(train_accuracies[2], val_accuracies[2], test_accuracies[2])


    # TODO: Train the classifier with different weighty decay and plot
    #pass
```

```
weigh_decay:0.000010
weigh_decay:0.001000
weigh_decay:1.000000
```







Inline Question 2.

Discuss underfitting and overfitting as observed in the 3 graphs obtained by changing the regularization. Which weight_decay term gave you the best classifier performance? HINT: Do not just think in terms of best training set performance, keep in mind that the real utility of a machine learning model is when it performs well on data it has never seen before

Your Answer:

The weight_decay of 0.0001 provides the best balance, showing minimal signs of overfitting (small gap between training and validation curves) while maintaining high performance on both validation and test sets. With stronger regularization (0.01,1), we observe some underfitting as both training and validation accuracies are lower, indicating the model is too constrained to learn effectively.

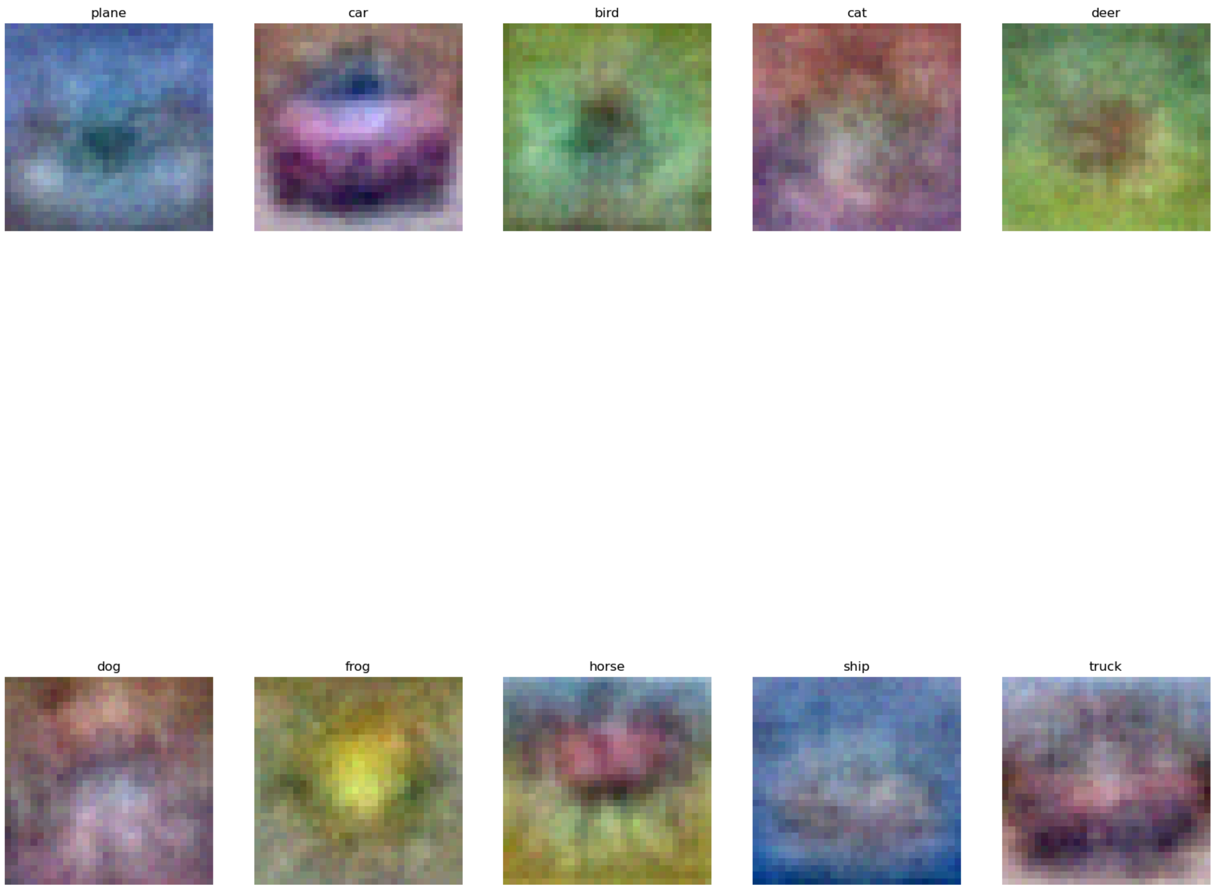
Visualize the filters (10%)

In [87]:

```
# These visualizations will only somewhat make sense if your learning rate a
# properly chosen in the model. Do your best.

# TODO: Run this cell and Show filter visualizations for the best set of wei
# Report the 2 hyperparameters you used to obtain the best model.

# NOTE: You need to set `best_learning_rate` and `best_weight_decay` to the
best_learning_rate = 0.001
best_weight_decay = 0.00001
print("Best LR:", best_learning_rate)
print("Best Weight Decay:", best_weight_decay)

train_accu, val_accu, test_accu, best_weights = train(best_learning_rate, be
# NOTE: You need to set `best_weights` to the weights with the highest accur
w = best_weights[:, :-1]
w = w.reshape(10, 3, 32, 32).transpose(0, 2, 3, 1)

w_min, w_max = np.min(w), np.max(w)

fig = plt.figure(figsize=(20, 20))
classes = [
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
for i in range(10):
    fig.add_subplot(2, 5, i + 1)
```

```
    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[i, :, :, :].squeeze() - w_min) / (w_max - w_min)
    # plt.imshow(wimg.astype('uint8'))
    plt.imshow(wimg.astype(int))
    plt.axis("off")
    plt.title(classes[i])
plt.show()
```

```
Best LR: 0.001
Best Weight Decay: 1e-05
```




```
In [ ]:
```