1. Using Convolutional Neural Network for Image Recognition ¶

In [32]:
```
import warnings
import os
warnings.filterwarnings("ignore")
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
from tensorflow.keras import utils, datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

import matplotlib as mpl
import pickle
import csv
import itertools
from collections import defaultdict
import time
import pandas as pd
import math
from tqdm import tqdm
!pip install dill
import dill
```

In [43]:
```
tf.test.is_gpu_available()
```

Out[43]: True

In [34]:
```
tf.config.list_physical_devices('GPU')
```

Out[34]: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

In [35]:
```
tf.__version__
```

Out[35]: '2.5.0'

In [333]:
```
import urllib.request
if not os.path.exists("lab11_1_lib.py"):
    urllib.request.urlretrieve("https://nthu-datalab.github.io/ml/labs/11-1_CNN/lab11_1_lib.py", "lab11_1_lib.py")

from lab11_1_lib import draw_timeline
```

import MNIST dataset from Tensorflow.

In [330]:
```
# Download and prepare the MNIST dataset
(train_image, train_label), (test_image, test_label) = datasets.mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_image, test_image = train_image / 255.0, test_image / 255.0
print('shape of train_image:', train_image.shape)
print('shape of train_label:', train_label.shape)
print('shape of test_image:', test_image.shape)
print('shape of test_label:', test_label.shape)
```

```
shape of train_image: (60000, 28, 28)
shape of train_label: (60000,)
shape of test_image: (10000, 28, 28)
shape of test_label: (10000,)
```

Multilayer Convolutional Network on MNIST

In [331]:
```
# reshaping the training data to 3 dimensions
train_image_2 = train_image.reshape((60000, 28, 28, 1))
test_image_2 = test_image.reshape((10000, 28, 28, 1))
print(train_image_2.shape)
print(test_image_2.shape)
```

```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
```

MNIST has one color channel.(because the image are grayscale)
In this example, we will configure our CNN to process inputs of shape (28, 28, 1), which is the format of MNIST image. We do this by passing the argument input_shape to our first layer.

1-1

Please implement a CNN for image recognition using the MNIST dataset. You have to design your network architecture and analyze the effect of different stride size and filter size. Also, plot the learning curve, accuracy of training and test sets, and distributions of weights and biases.

model_1

data spilt: 55, 000 examples are in the training set, 5, 000 in the validation set, and 10, 000 in the test set. model setting:

- Epochs = 100
- batch size = 5000
- strides=(1,1),filter=(3,3)
- activation='relu','softmax'(output)
- Optimizer = Adam

In [332]:
```python
#The convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model_1 = models.Sequential()
model_1.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='same', activation='relu', input_shape=(28, 28, 1)))
model_1.add(layers.MaxPooling2D((2, 2)))
model_1.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
model_1.add(layers.MaxPooling2D((2, 2)))
model_1.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
model_1.add(layers.Flatten())
model_1.add(layers.Dense(64, activation='relu'))
model_1.add(layers.Dropout(0.5))
model_1.add(layers.Dense(10, activation='softmax'))
model_1.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_45 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
flatten_16 (Flatten)         (None, 3136)              0
_________________________________________________________________
dense_41 (Dense)             (None, 64)                200768
_________________________________________________________________
dropout_25 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_42 (Dense)             (None, 10)                650
=================================================================
Total params: 257,162
Trainable params: 257,162
Non-trainable params: 0
_________________________________________________________________
```

In [334]:
```python
model_1.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [335]:
```python
from tensorflow.keras.callbacks import Callback

# 回调函数定义
class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback = TestAccuracyCallback(test_data=(test_image_2, test_label))
```

In [336]:

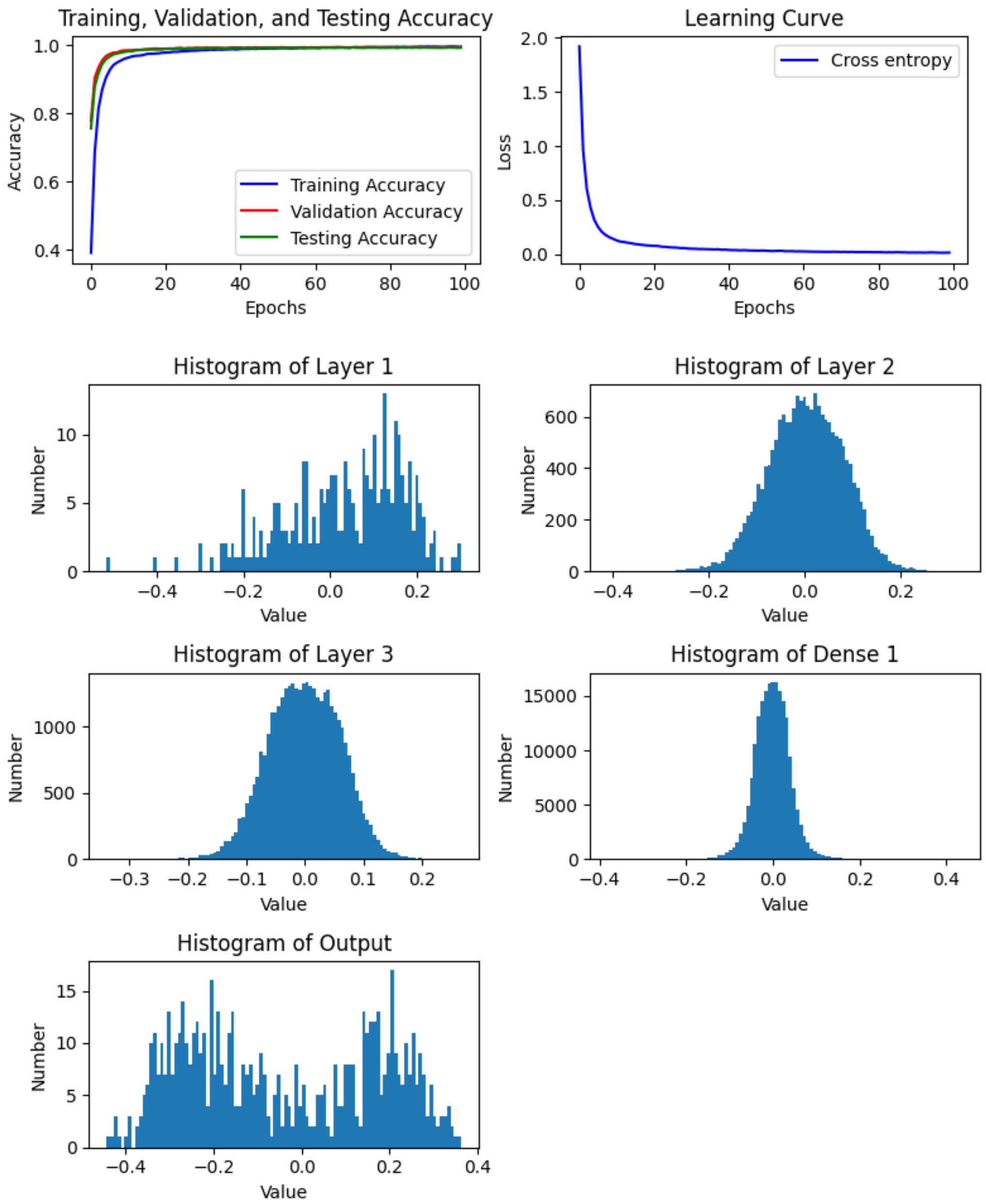
```
np.random.seed(111024520)
history = model_1.fit(train_image_2, train_label , batch_size=5000, epochs=100, validation_split=1/12, callbacks=[test_callback])
```

```
Epoch 96/100
11/11 [==============================] - 0s 34ms/step - loss: 0.0136 - accuracy: 0.9956 - val_loss: 0.0350 - val_accuracy: 0.9938

Test accuracy after epoch 96: 0.9927
Epoch 97/100
11/11 [==============================] - 0s 33ms/step - loss: 0.0119 - accuracy: 0.9963 - val_loss: 0.0353 - val_accuracy: 0.9942

Test accuracy after epoch 97: 0.9936
Epoch 98/100
11/11 [==============================] - 0s 34ms/step - loss: 0.0115 - accuracy: 0.9965 - val_loss: 0.0344 - val_accuracy: 0.9948

Test accuracy after epoch 98: 0.9931
Epoch 99/100
11/11 [==============================] - 0s 32ms/step - loss: 0.0119 - accuracy: 0.9957 - val_loss: 0.0359 - val_accuracy: 0.9944

Test accuracy after epoch 99: 0.9923
Epoch 100/100
11/11 [==============================] - 0s 33ms/step - loss: 0.0127 - accuracy: 0.9955 - val_loss: 0.0359 - val_accuracy: 0.9944

Test accuracy after epoch 100: 0.9929
```

In [337]:

```python
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.15,7/3))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights = []

for layer in model_1.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 3:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
    plt.show()
```

```
plot_weights_histogram(all_weights)

plt.show()
```



From the above plots, we can find that:

1. After 20 epochs, the accuracy rates and learning rate tend to stabilize.
2. The distribution of the weights in layer2,3, dense1 looks like a normal distribution.

model_2

Then, we try to set different stride size= $(2, 2)$.

- Epochs = 100
- batch size = 5000
- strides=(2,2),filter=(3,3)
- activation='relu','softmax'(output)
- Optimizer = Adam

In [338]:

```
#The convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model_2 = models.Sequential()
model_2.add(layers.Conv2D(32, (3, 3), strides=(2,2), padding='same', activation='relu', input_shape=(28, 28, 1)))
model_2.add(layers.MaxPooling2D((2, 2)))
model_2.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='same', activation='relu'))
model_2.add(layers.MaxPooling2D((2, 2)))
model_2.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='same', activation='relu'))
model_2.add(layers.Flatten())
model_2.add(layers.Dense(64, activation='relu'))
model_2.add(layers.Dropout(0.5))
model_2.add(layers.Dense(10, activation='softmax'))
model_2.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_48 (Conv2D)           (None, 14, 14, 32)        320       
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 7, 7, 32)          0         
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 4, 4, 64)          18496     
_________________________________________________________________
max_pooling2d_41 (MaxPooling (None, 2, 2, 64)          0         
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 1, 1, 64)          36928     
_________________________________________________________________
flatten_17 (Flatten)         (None, 64)                0         
_________________________________________________________________
dense_43 (Dense)             (None, 64)                4160      
_________________________________________________________________
dropout_26 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_44 (Dense)             (None, 10)                650       
=================================================================
Total params: 60,554
Trainable params: 60,554
Non-trainable params: 0
_________________________________________________________________
```

In [339]:
```python
model_2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

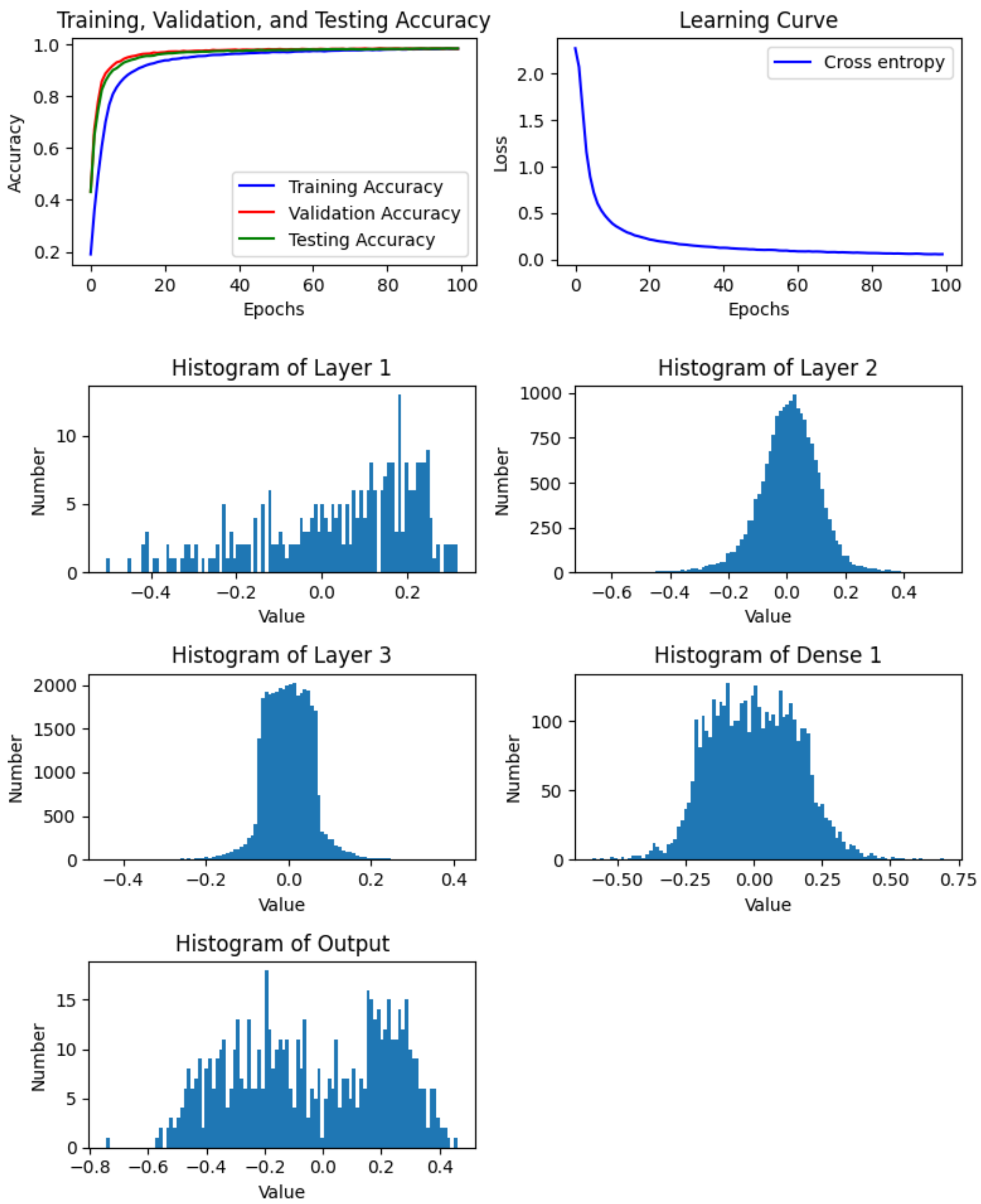
In [340]:
```python
from tensorflow.keras.callbacks import Callback

# 回调函数定义
class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback2 = TestAccuracyCallback(test_data=(test_image_2, test_label))
```

In [341]:
```python
np.random.seed(111024520)

history_2 = model_2.fit(train_image_2, train_label, batch_size=5000, epochs=100, validation_split=1/12, callbacks=[test_callback2])
```

```
Epoch 96/100
11/11 [==============================] - 0s 14ms/step - loss: 0.0553 - accuracy: 0.9844 - val_loss: 0.0487 - val_accuracy: 0.9844

Test accuracy after epoch 96: 0.9853
Epoch 97/100
11/11 [==============================] - 0s 21ms/step - loss: 0.0551 - accuracy: 0.9848 - val_loss: 0.0469 - val_accuracy: 0.9860

Test accuracy after epoch 97: 0.9856
Epoch 98/100
11/11 [==============================] - 0s 19ms/step - loss: 0.0562 - accuracy: 0.9840 - val_loss: 0.0461 - val_accuracy: 0.9854

Test accuracy after epoch 98: 0.9859
Epoch 99/100
11/11 [==============================] - 0s 15ms/step - loss: 0.0545 - accuracy: 0.9847 - val_loss: 0.0477 - val_accuracy: 0.9850

Test accuracy after epoch 99: 0.9850
Epoch 100/100
11/11 [==============================] - 0s 14ms/step - loss: 0.0551 - accuracy: 0.9842 - val_loss: 0.0473 - val_accuracy: 0.9846

Test accuracy after epoch 100: 0.9857
```

In [342]:
```python
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.1,7/3))
plt.subplot(1, 2, 1)
plt.plot(history_2.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history_2.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback2.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history_2.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights_2 = []

for layer in model_2.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights_2.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 3:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
    plt.show()
```

```
plot_weights_histogram(all_weights_2)

plt.show()
```



From the above plot, we can find that when we increse thestride size from (1,1) to (2,2), the results are similar to the previous model.

model_3

Then, we try to set different filter size= $(5,5)$.

- Epochs = 100
- batch size = 5000
- strides=(1,1),filter=(5,5)
- activation='relu','softmax'(output)
- Optimizer = Adam

In [343]:
```
#The convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model_3 = models.Sequential()
model_3.add(layers.Conv2D(32, (5, 5), strides=(1,1), padding='same', activation='relu', input_shape=(28, 28, 1)))
model_3.add(layers.MaxPooling2D((2, 2)))
model_3.add(layers.Conv2D(64, (5, 5), strides=(1,1), padding='same', activation='relu'))
model_3.add(layers.MaxPooling2D((2, 2)))
model_3.add(layers.Conv2D(64, (5, 5), strides=(1,1), padding='same', activation='relu'))
model_3.add(layers.Flatten())
model_3.add(layers.Dense(64, activation='relu'))
model_3.add(layers.Dropout(0.5))
model_3.add(layers.Dense(10, activation='softmax'))
model_3.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_51 (Conv2D)           (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 14, 14, 64)        51264     
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 7, 7, 64)          0         
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 7, 7, 64)          102464    
_________________________________________________________________
flatten_18 (Flatten)         (None, 3136)              0         
_________________________________________________________________
dense_45 (Dense)             (None, 64)                200768    
_________________________________________________________________
dropout_27 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_46 (Dense)             (None, 10)                650       
=================================================================
Total params: 355,978
Trainable params: 355,978
Non-trainable params: 0
_________________________________________________________________
```

In [344]:
```
model_3.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [345]:

```python
from tensorflow.keras.callbacks import Callback

# 回调函数定义
class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback3 = TestAccuracyCallback(test_data=(test_image_2, test_label))
```

In [346]:

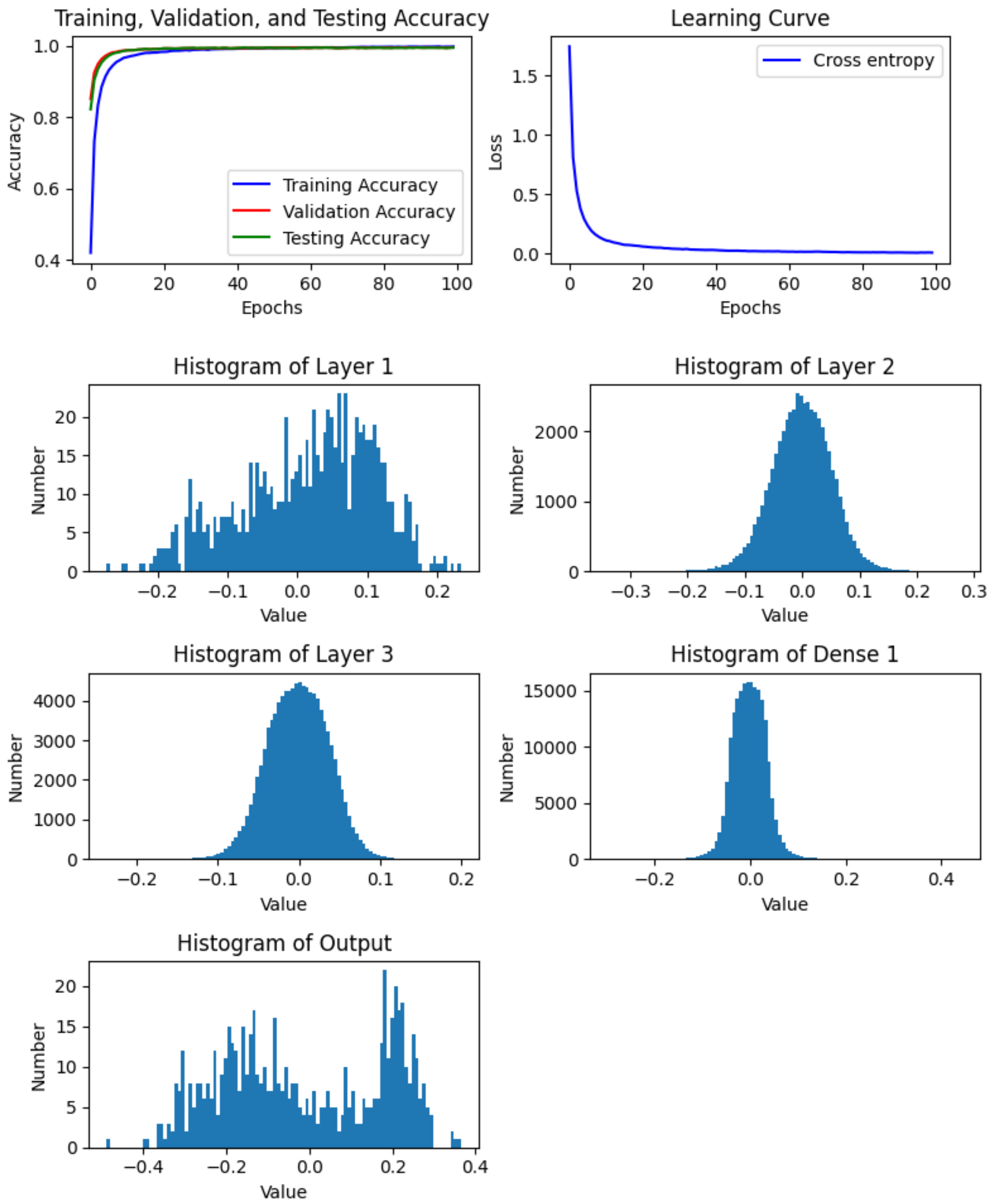
```python
np.random.seed(111024520)

history_3 = model_3.fit(train_image_2, train_label, batch_size=5000, epochs=100, validation_split=1/12, callbacks=[test_callback3])
```

```
Epoch 96/100
11/11 [==============================] - 1s 47ms/step - loss: 0.0076 - accuracy: 0.9975 - val_loss: 0.0354 - val_accuracy: 0.9944

Test accuracy after epoch 96: 0.9935
Epoch 97/100
11/11 [==============================] - 0s 45ms/step - loss: 0.0097 - accuracy: 0.9962 - val_loss: 0.0352 - val_accuracy: 0.9946

Test accuracy after epoch 97: 0.9938
Epoch 98/100
11/11 [==============================] - 0s 46ms/step - loss: 0.0088 - accuracy: 0.9967 - val_loss: 0.0359 - val_accuracy: 0.9934

Test accuracy after epoch 98: 0.9931
Epoch 99/100
11/11 [==============================] - 0s 46ms/step - loss: 0.0100 - accuracy: 0.9966 - val_loss: 0.0330 - val_accuracy: 0.9942

Test accuracy after epoch 99: 0.9938
Epoch 100/100
11/11 [==============================] - 0s 44ms/step - loss: 0.0086 - accuracy: 0.9972 - val_loss: 0.0308 - val_accuracy: 0.9948

Test accuracy after epoch 100: 0.9941
```

In [347]:

```
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(8.97,7/3))
plt.subplot(1, 2, 1)
plt.plot(history_3.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history_3.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback3.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history_3.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights_3 = []

for layer in model_3.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights_3.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 3:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
    plt.show()
```

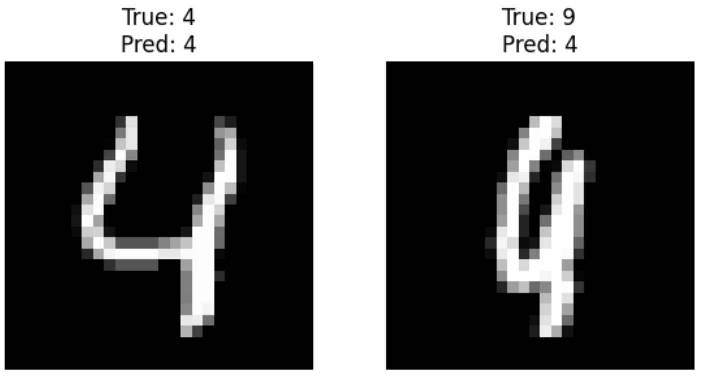
```
plot_weights_histogram(all_weights_3)

plt.show()
```



From the above plot, we can find that when we increse the filter size from (3,3) to (5,5), the results are similar to the previous model.

1-2 Show some examples of correctly classified and miss-classified images and discuss your results.

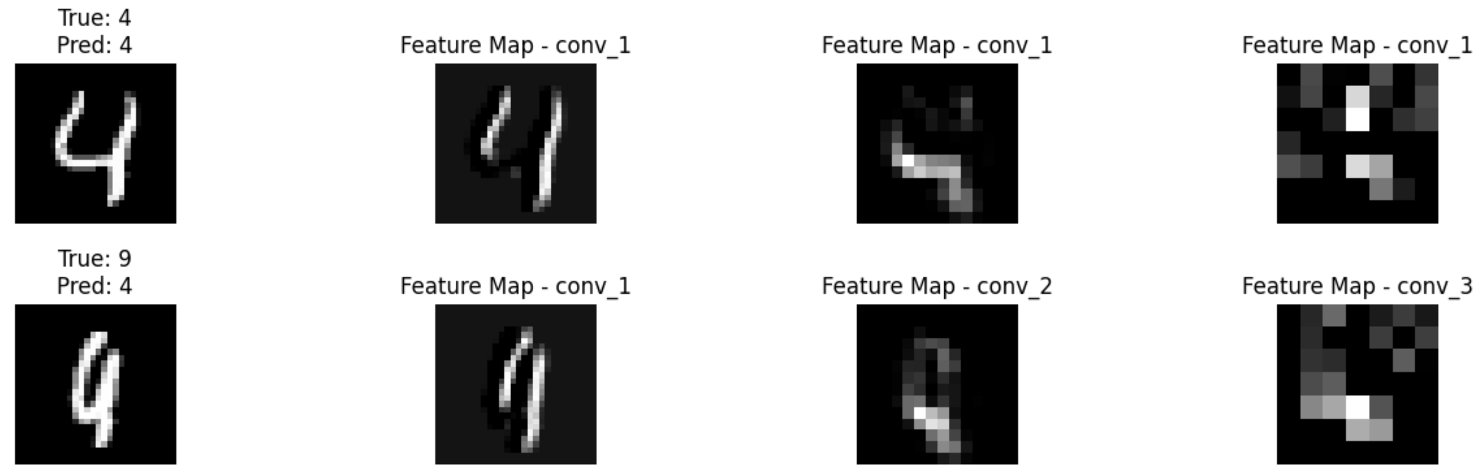
In [348]:
```
predictions = model_1.predict(test_image_2)
predicted_labels = np.argmax(predictions, axis=1) #10000個

# true_and_pred_4
true_and_pred_4_indices = np.where((test_label == 4) & (predicted_labels == 4))[0]

# true_not_4_pred_4
true_not_4_pred_4_indices = np.where((test_label != 4) & (predicted_labels == 4))[0]

# true_and_pred_4 第一張圖
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plt.imshow(test_image[true_and_pred_4_indices[0]], cmap='gray')
plt.title(f'True: {test_label[true_and_pred_4_indices[0]]}\nPred: {predicted_labels[true_and_pred_4_indices[0]]}')
plt.axis('off')

# true_not_4_pred_4 第一張圖
plt.subplot(1, 2, 2)
plt.imshow(test_image[true_not_4_pred_4_indices[0]], cmap='gray')
plt.title(f'True: {test_label[true_not_4_pred_4_indices[0]]}\nPred: {predicted_labels[true_not_4_pred_4_indices[0]]}')
plt.axis('off')

plt.tight_layout()
plt.show()
```



The above plots are examples of correctly classified and mis-classified images.
It show a true label and predicted label of 4 and another example where the true label is not 4 but the predicted label is 4.
I think the pattern of right image looks like 4, so it was miss-classified.

1-3 Following 1-2, observe the feature maps from different convolutional layers and describe how a feature map changes with increasing depth.

In [349]:
```python
# Choose an input image from the test set
input_image_1 = test_image_2[true_and_pred_4_indices[0]].reshape(1, 28, 28, 1)
input_image_2 = test_image_2[true_not_4_pred_4_indices[0]].reshape(1, 28, 28, 1)

# Get feature maps for the last convolutional layer in each block
ixs = [0, 2, 4]
outputs = [model_1.layers[i].output for i in ixs]
model = tf.keras.Model(inputs=model_1.input, outputs=outputs)
feature_maps_1 = model.predict(input_image_1)
feature_maps_2 = model.predict(input_image_2)
layer_names = [layer.name for layer in model_1.layers[:5] if 'conv2d' in layer.name ]

# Plot the feature maps
plt.figure(figsize=(16, 4))

# Plot the original images for both cases
plt.subplot(2, 4, 1)
plt.imshow(test_image[true_and_pred_4_indices[0]], cmap='gray')
plt.title(f'True: {test_label[true_and_pred_4_indices[0]]}\nPred: {predicted_labels[true_and_pred_4_indices[0]]}')
plt.axis('off')

plt.subplot(2, 4, 5)
plt.imshow(test_image[true_not_4_pred_4_indices[0]], cmap='gray')
plt.title(f'\nTrue: {test_label[true_not_4_pred_4_indices[0]]}\nPred: {predicted_labels[true_not_4_pred_4_indices[0]]}')
plt.axis('off')

j=1
# Plot the feature maps for each block for the first case
for i, fmap in enumerate(feature_maps_1):
    plt.subplot(2, 4, i + 2)  # i + 2 because the original image is at position 1
    plt.imshow(fmap[0, :, :, 0], cmap='gray')  # Assuming the last dimension represents the number of filters
    plt.title(f'Feature Map - conv_{j}')
    plt.axis('off')

j=1
# Plot the feature maps for each block for the second case
for i, fmap in enumerate(feature_maps_2):
    plt.subplot(2, 4, i + 6)  # i + 6 because the original image is at position 5
    plt.imshow(fmap[0, :, :, 0], cmap='gray')  # Assuming the last dimension represents the number of filters
    plt.title(f'Feature Map - conv_{j}')
    j+=1
    plt.axis('off')

# Adjust the separation between rows
plt.subplots_adjust(hspace=0.5)
# Show the figure
plt.show()
```



From the above plot, we can find that:

1. Feature maps closer to the input of the model capture a lot of fine detail in the image and that as we progress deeper into the model, the feature maps show less and less detail.
2. This pattern was to be expected, as the model abstracts the features from the image into more general concepts that can be used to make a classification.

1-4 Following 1-1, please add L2 regularization to the CNN implemented in 1-1 and discuss its effect.

We set $\alpha = 0.05$ of L2 regularization.

- Epochs = 100
- batch size = 5000
- strides=(1,1),filter=(3,3)
- kernel_regularizer=0.05
- activation='relu','softmax'(output)
- Optimizer = Adam

In [350]:

```python
from tensorflow.keras import layers, models, regularizers

# Build the CNN model with L2 regularization
model_L2 = models.Sequential()
model_L2.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='same', activation='relu', kernel_regularizer=regularizers.l2(0.05), input_shape=(28, 28, 1)))
model_L2.add(layers.MaxPooling2D((2, 2)))
model_L2.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu', kernel_regularizer=regularizers.l2(0.05)))
model_L2.add(layers.MaxPooling2D((2, 2)))
model_L2.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu', kernel_regularizer=regularizers.l2(0.05)))

model_L2.add(layers.Flatten())
model_L2.add(layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.05)))
model_L2.add(layers.Dropout(0.5))
model_L2.add(layers.Dense(10, activation='softmax', kernel_regularizer=regularizers.l2(0.05)))
model_L2.summary()
# Compile the model
model_L2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_54 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
flatten_19 (Flatten)         (None, 3136)              0
_________________________________________________________________
dense_47 (Dense)             (None, 64)                200768
_________________________________________________________________
dropout_28 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_48 (Dense)             (None, 10)                650
=================================================================
Total params: 257,162
Trainable params: 257,162
Non-trainable params: 0
_________________________________________________________________
```

In [351]:
```python
from tensorflow.keras.callbacks import Callback

class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback4 = TestAccuracyCallback(test_data=(test_image_2, test_label))
```

In [352]:
```python
np.random.seed(111024520)

history_L2 = model_L2.fit(train_image_2, train_label, batch_size=5000, epochs=100, validation_split=1/12, callbacks=[test_callback4])
```

```
11/11 [==============================] - 0s 32ms/step - loss: 1.2205 - accuracy: 0.8407 - val_loss: 1.0116 - val_accuracy: 0.9198

Test accuracy after epoch 96: 0.8954
Epoch 97/100
11/11 [==============================] - 0s 36ms/step - loss: 1.2250 - accuracy: 0.8385 - val_loss: 1.0110 - val_accuracy: 0.9200

Test accuracy after epoch 97: 0.8969
Epoch 98/100
11/11 [==============================] - 0s 33ms/step - loss: 1.2224 - accuracy: 0.8399 - val_loss: 1.0065 - val_accuracy: 0.9182

Test accuracy after epoch 98: 0.8967
Epoch 99/100
11/11 [==============================] - 0s 32ms/step - loss: 1.2215 - accuracy: 0.8407 - val_loss: 1.0099 - val_accuracy: 0.9196

Test accuracy after epoch 99: 0.8962
Epoch 100/100
11/11 [==============================] - 0s 33ms/step - loss: 1.2238 - accuracy: 0.8380 - val_loss: 1.0027 - val_accuracy: 0.9220

Test accuracy after epoch 100: 0.8981
```

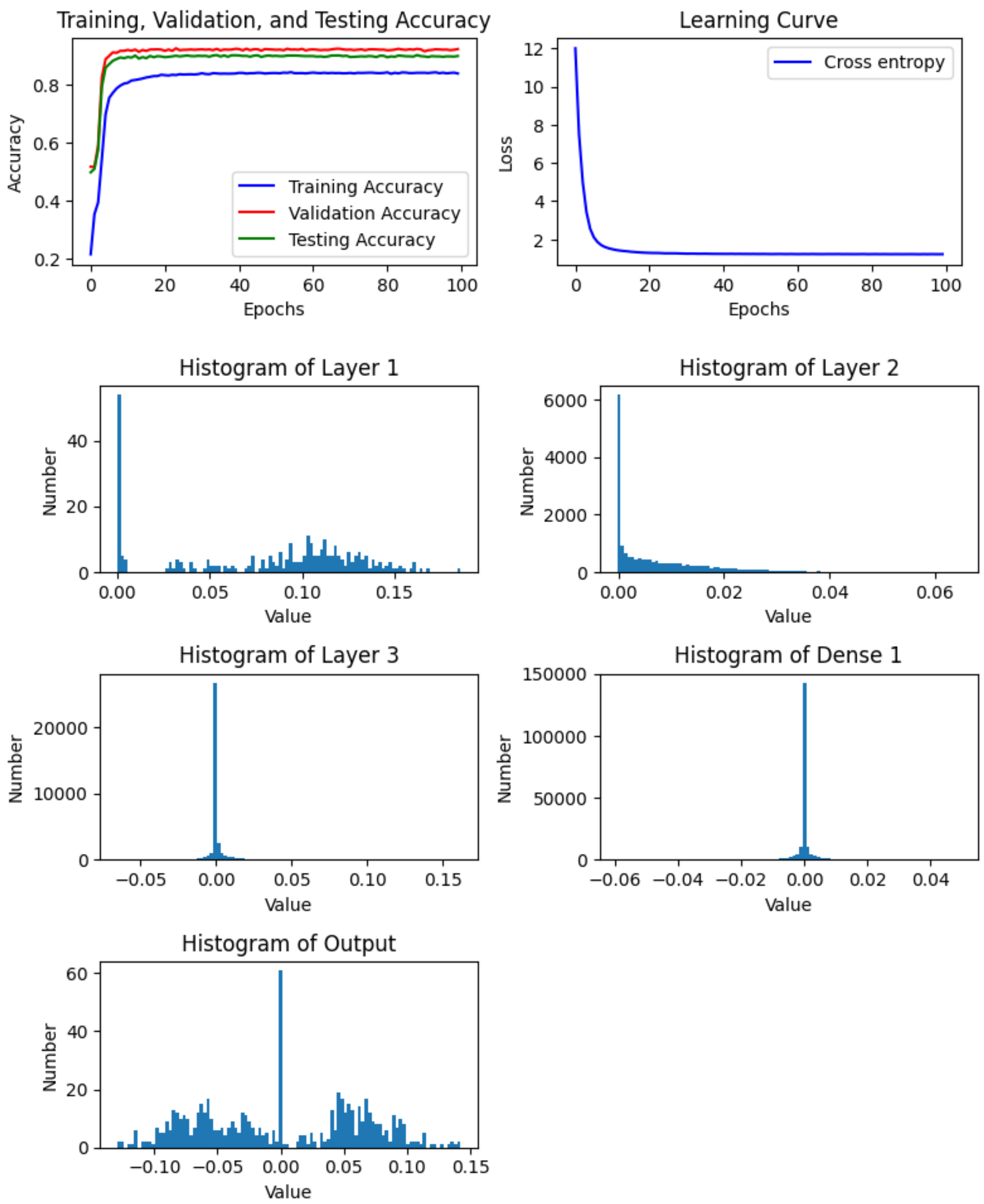
In [353]:
```python
_, test_acc_L2 = model_L2.evaluate(test_image_2, test_label, verbose=0)
print('Testing Accuracy : %.4f'%test_acc_L2)
```

```
Testing Accuracy : 0.8981
```

In [354]:

```
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.1,7/3))
plt.subplot(1, 2, 1)
plt.plot(history_L2.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history_L2.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback4.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history_L2.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights_4 = []

for layer in model_L2.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights_4.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 3:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
    plt.show()
```

```
plot_weights_histogram(all_weights_4)

plt.show()
```



We can find that:

1. The histograms of weights appear to be concentrated around a narrow range, and the value appear to 0.
2. The regularization term encourages the network to learn smaller weight values.

2 Preprocessing Before Using Convolutional Neural Network for Image Recognition

2-5 Data preprocessing

- data: a 10000x3072 numpy array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.
- labels: a list of 10000 numbers in the range 0-9. The number at index i indicates the label of the ith image in the array data.

1. We first reshape the batches of data to be $(10000, 3, 32, 32)$, and then combined 5 batches to be a train set image with dim= $(50000, 32, 32, 3)$.
2. Similarly, we reshape the batches of label and combined them to be to be a train set label with dim= $(50000, 1)$
3. Reshape test set image to be dim= $(10000, 3, 32, 32)$.
4. Reshape test set label to be dim= $(10000, 1)$.

In [85]:
```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

In [90]:
```
batch = unpickle("C:/Users/cluster/Desktop/lee/HW2/batches.meta")
```

In [236]:
```
label_names = batch[b'label_names']
label_names = [label.decode('utf-8') for label in label_names]
label_names
```

Out[236]:
```
['airplane',
 'automobile',
 'bird',
 'cat',
 'deer',
 'dog',
 'frog',
 'horse',
 'ship',
 'truck']
```

In [91]:
```
data1 = unpickle("C:/Users/cluster/Desktop/lee/HW2/data_batch_1")
data1, label1 = data1[b'data'], data1[b'labels'] #10000x3072, 10000
data2 = unpickle("C:/Users/cluster/Desktop/lee/HW2/data_batch_2")
data2, label2 = data2[b'data'], data2[b'labels']
data3 = unpickle("C:/Users/cluster/Desktop/lee/HW2/data_batch_3")
data3, label3 = data3[b'data'], data3[b'labels']
data4 = unpickle("C:/Users/cluster/Desktop/lee/HW2/data_batch_4")
data4, label4 = data4[b'data'], data4[b'labels']
data5 = unpickle("C:/Users/cluster/Desktop/lee/HW2/data_batch_5")
data5, label5 = data5[b'data'], data5[b'labels']
```

In [194]:
```
data1_new = data1.reshape(10000,3,32,32) #每個圖像的通道維度（紅、綠、藍）被分開。
data2_new = data2.reshape(10000,3,32,32)
data3_new = data3.reshape(10000,3,32,32)
data4_new = data4.reshape(10000,3,32,32)
data5_new = data5.reshape(10000,3,32,32)
```

In [195]:
```
train_image2 = np.concatenate((data1_new, data2_new, data3_new, data4_new, data5_new), axis=0)
train_image2=np.transpose(train_image2, (0,2,3,1))
print(train_image2.shape)
```

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

In [204]:
```
#label1是list轉成array
train_label2 = np.concatenate((np.array(label1),np.array(label2),np.array(label3),np.array(label4),np.array(label5)))
train_label2=train_label2.reshape(50000,)
```

In [205]:
```
data_test = unpickle("C:/Users/cluster/Desktop/lee/HW2/test_batch")
test_image2, test_label2 = data_test[b'data'], data_test[b'labels']
```

In [206]:
```
test_image2 = test_image2.reshape(10000,3,32,32)
test_image2 = np.transpose(test_image2, (0,2,3,1))
test_label2 = (np.array(test_label2)).reshape(10000,)
```

In [232]:
```
test_label2
```

Out[232]:
```
array([3, 8, 8, ..., 5, 1, 7])
```

In [207]:
```
print('X_train shape:', train_image2.shape)
print('Y_train shape:', train_label2.shape)
print('X_test shape:', test_image2.shape)
print('Y_test shape:', test_label2.shape)
```

```
X_train shape: (50000, 32, 32, 3)
Y_train shape: (50000,)
X_test shape: (10000, 32, 32, 3)
Y_test shape: (10000,)
```

In [237]:

```
plt.figure(figsize=(6, 3))
plt.imshow(train_image2[150], cmap='gray')
plt.title(f'True: {label_names[train_label2[150]]}')
plt.axis('off')
plt.xticks([])
plt.show()
```

True: cat

2-1 CNN on CIFAR-10

model 2_1

- Epochs = 200
- batch size = 500
- strides=(3,3),filter=(5,5)
- activation='relu','softmax'(output)
- Optimizer = Adam

In [120]:

```
model2_1 = models.Sequential()

#The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model2_1.add(layers.Conv2D(64, (5, 5), strides=(3,3), padding='same', activation='relu', input_shape=(32, 32, 3)))
model2_1.add(layers.MaxPool2D(((2, 2))))
model2_1.add(layers.Conv2D(64, (5, 5), strides=(3,3),padding='same', activation='relu'))
model2_1.add(layers.MaxPool2D(((2, 2))))

model2_1.add(layers.Flatten())

model2_1.add(layers.Dense(384, activation='relu'))
model2_1.add(layers.Dropout(0.2))
model2_1.add(layers.Dense(192, activation='relu'))
model2_1.add(layers.Dropout(0.2))
model2_1.add(layers.Dense(10, activation='softmax'))
model2_1.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model2_1.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_29 (Conv2D)           (None, 11, 11, 64)        4864      
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 2, 2, 64)          102464    
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 1, 1, 64)          0         
_________________________________________________________________
flatten_11 (Flatten)         (None, 64)                0         
_________________________________________________________________
dense_26 (Dense)             (None, 384)               24960     
_________________________________________________________________
dropout_15 (Dropout)         (None, 384)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 192)               73920     
_________________________________________________________________
dropout_16 (Dropout)         (None, 192)               0         
_________________________________________________________________
dense_28 (Dense)             (None, 10)                1930      
=================================================================
Total params: 208,138
Trainable params: 208,138
Non-trainable params: 0
_________________________________________________________________
```

In [121]:

```python
from tensorflow.keras.callbacks import Callback

class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback2_1 = TestAccuracyCallback(test_data=(test_image2, test_label2))
```

In [122]:

```python
history2_1 = model2_1.fit(train_image2, train_label2, batch_size=500, epochs=200, validation_split=5000/50000, callbacks=[test_callback2_1])
```

```
Epoch 196/200
90/90 [==============================] - 0s 5ms/step - loss: 0.5648 - accuracy: 0.7940 - val_loss: 2.7030 - val_accuracy: 0.4856

Test accuracy after epoch 196: 0.4841
Epoch 197/200
90/90 [==============================] - 1s 6ms/step - loss: 0.5640 - accuracy: 0.7925 - val_loss: 2.6801 - val_accuracy: 0.4888

Test accuracy after epoch 197: 0.4715
Epoch 198/200
90/90 [==============================] - 0s 6ms/step - loss: 0.5776 - accuracy: 0.7898 - val_loss: 2.5868 - val_accuracy: 0.4834

Test accuracy after epoch 198: 0.4755
Epoch 199/200
90/90 [==============================] - 1s 6ms/step - loss: 0.5645 - accuracy: 0.7946 - val_loss: 2.6699 - val_accuracy: 0.4906

Test accuracy after epoch 199: 0.4782
Epoch 200/200
90/90 [==============================] - 0s 5ms/step - loss: 0.5616 - accuracy: 0.7946 - val_loss: 2.7053 - val_accuracy: 0.4860

Test accuracy after epoch 200: 0.4757
```

In [123]:

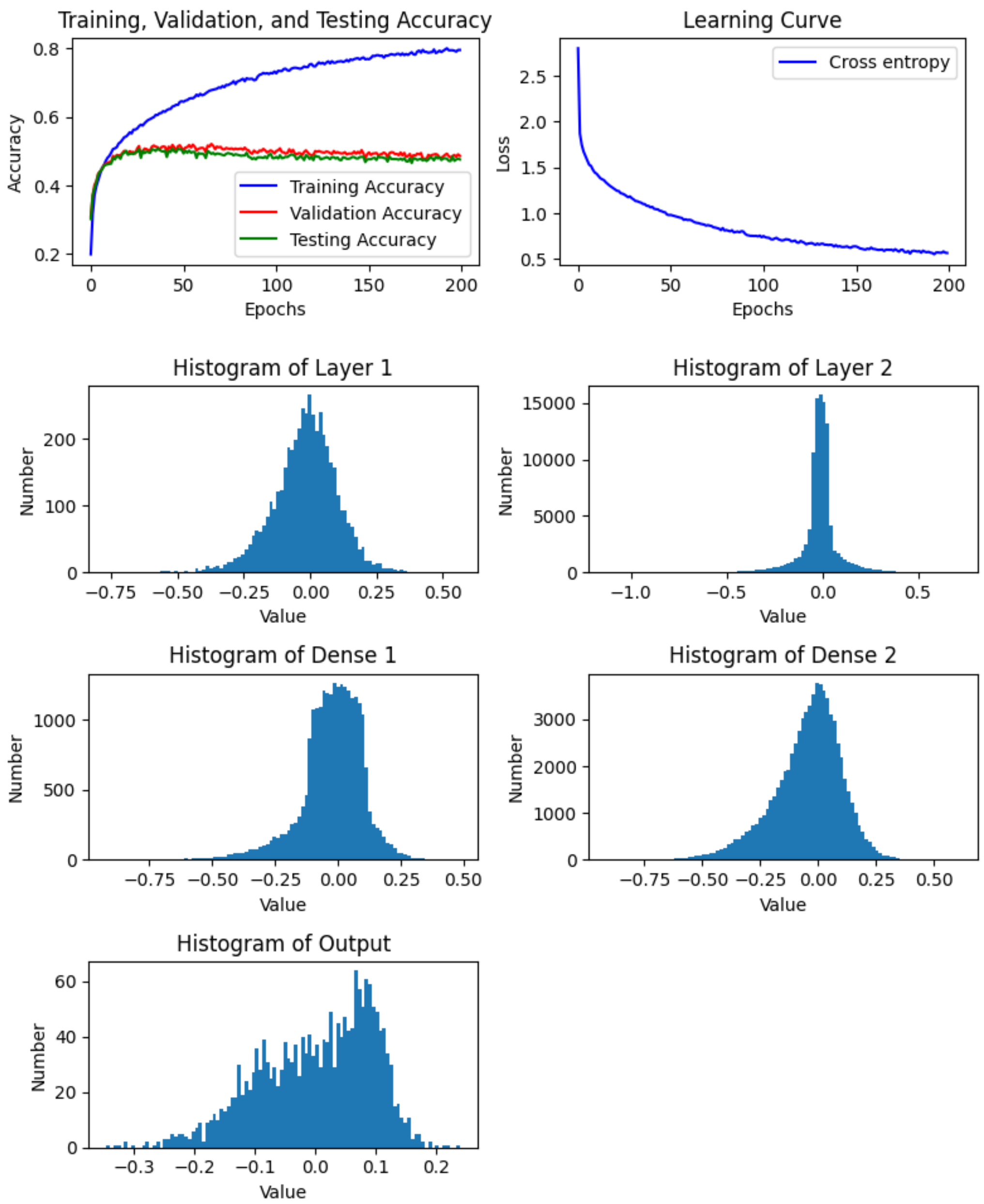
```python
model2_1.layers
```

Out[123]:

```
[<tensorflow.python.keras.layers.convolutional.Conv2D at 0x1e6ab096f70>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x1e6aeda7730>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x1e6ab7158e0>,
 <tensorflow.python.keras.layers.pooling.MaxPooling2D at 0x1e6ab9f08b0>,
 <tensorflow.python.keras.layers.core.Flatten at 0x1e6ab744d00>,
 <tensorflow.python.keras.layers.core.Dense at 0x1e6aaf975b0>,
 <tensorflow.python.keras.layers.core.Dropout at 0x1e6b0973250>,
 <tensorflow.python.keras.layers.core.Dense at 0x1e6ab6734c0>,
 <tensorflow.python.keras.layers.core.Dropout at 0x1e6ae3d8370>,
 <tensorflow.python.keras.layers.core.Dense at 0x1e6ab748190>]
```

In [124]:
```
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.15,7/3))
plt.subplot(1, 2, 1)
plt.plot(history2_1.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history2_1.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback2_1.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history2_1.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights2_1 = []

for layer in model2_1.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights2_1.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 2:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 3:
            axes[row, col].set_title(f'Histogram of Dense 2')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
```

```
    plt.show()

plot_weights_histogram(all_weights2_1)

plt.show()
```



From the above plots, we can find that:

1. After 50 epochs, the validation accuracy and testing accuracy rates tend to stabilize around 0.5.
2. The learning rate decreases slowly.
3. The distribution of the weights in layers looks like a normal distribution.

model2_2

Then, we try to set different stride size$= (1, 1)$.

- Epochs = 200
- batch size = 500
- strides=(1,1),filter=(5,5)
- activation='relu','softmax'(output)
- Optimizer = Adam

In [130]:

```python
model2_2 = models.Sequential()

#The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model2_2.add(layers.Conv2D(64, (5, 5), strides=(1,1), padding='same', activation='relu', input_shape=(32, 32, 3)))
model2_2.add(layers.MaxPool2D(((2, 2))))
model2_2.add(layers.Conv2D(64, (5, 5), strides=(1,1),padding='same', activation='relu'))
model2_2.add(layers.MaxPool2D(((2, 2))))

model2_2.add(layers.Flatten())
model2_2.add(layers.Dense(384, activation='relu'))
model2_2.add(layers.Dropout(0.2))
model2_2.add(layers.Dense(192, activation='relu'))
model2_2.add(layers.Dropout(0.2))
model2_2.add(layers.Dense(10, activation='softmax'))
model2_2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model2_2.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_39 (Conv2D)           (None, 32, 32, 64)        4864      
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 16, 16, 64)        102464    
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
flatten_13 (Flatten)         (None, 4096)              0         
_________________________________________________________________
dense_32 (Dense)             (None, 384)               1573248   
_________________________________________________________________
dropout_19 (Dropout)         (None, 384)               0         
_________________________________________________________________
dense_33 (Dense)             (None, 192)               73920     
_________________________________________________________________
dropout_20 (Dropout)         (None, 192)               0         
_________________________________________________________________
dense_34 (Dense)             (None, 10)                1930      
=================================================================
Total params: 1,756,426
Trainable params: 1,756,426
Non-trainable params: 0
_________________________________________________________________
```

In [131]:
```
from tensorflow.keras.callbacks import Callback

class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback2_2 = TestAccuracyCallback(test_data=(test_image2, test_label2))
```

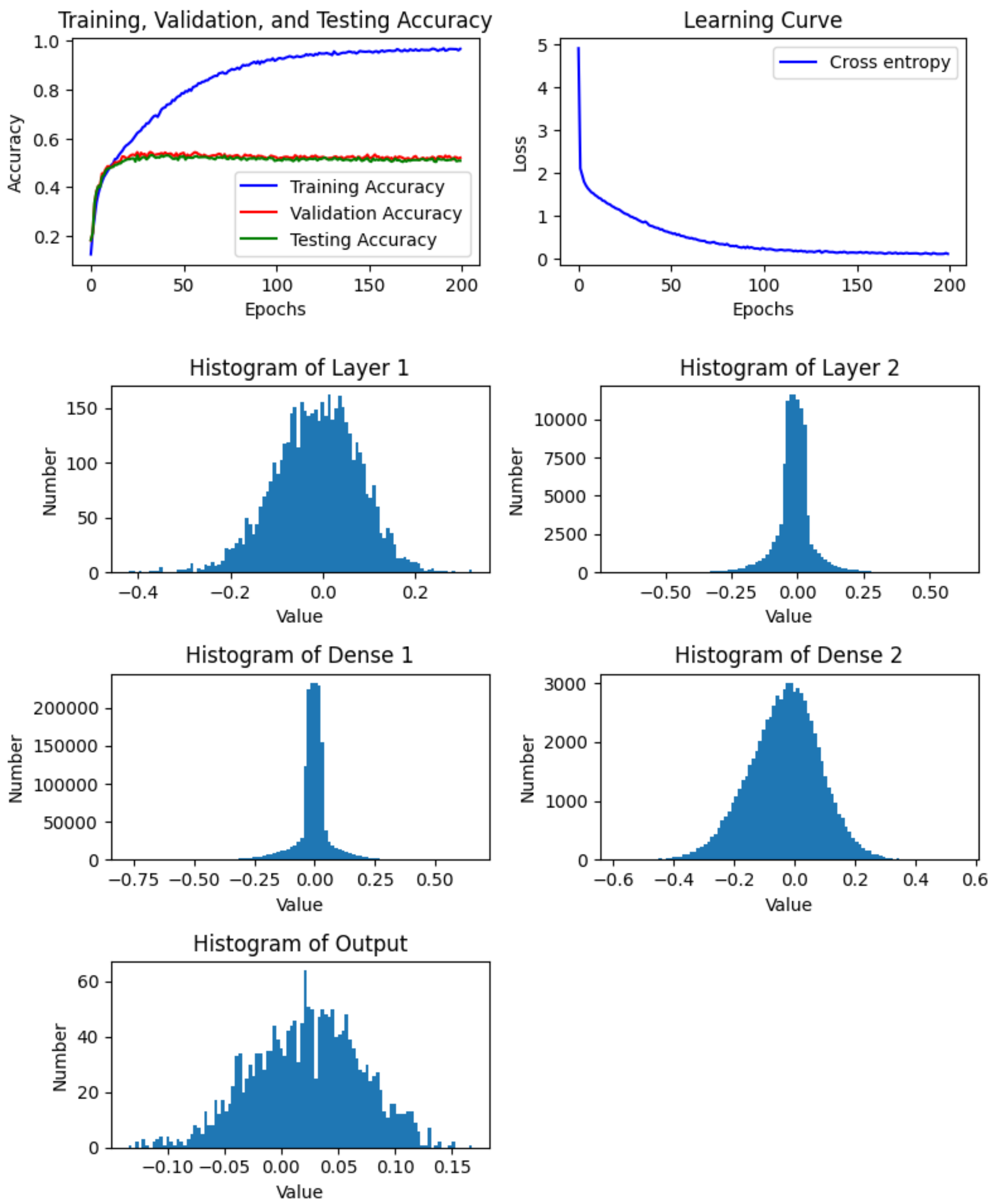
In [132]:
```
history2_2 = model2_2.fit(train_image2, train_label2, batch_size=500, epochs=200, validation_split=5000/50000, callbacks=[test_callback2_2])
```

```
90/90 [==============================] - 1s 8ms/step - loss: 0.1058 - accuracy: 0.9659 - val_loss: 3.8059 - val_accuracy: 0.5136

Test accuracy after epoch 196: 0.5170
Epoch 197/200
90/90 [==============================] - 1s 8ms/step - loss: 0.1062 - accuracy: 0.9657 - val_loss: 3.7413 - val_accuracy: 0.5240

Test accuracy after epoch 197: 0.5163
Epoch 198/200
90/90 [==============================] - 1s 8ms/step - loss: 0.1098 - accuracy: 0.9657 - val_loss: 3.7392 - val_accuracy: 0.5228

Test accuracy after epoch 198: 0.5069
Epoch 199/200
90/90 [==============================] - 1s 8ms/step - loss: 0.1266 - accuracy: 0.9609 - val_loss: 3.7791 - val_accuracy: 0.5172

Test accuracy after epoch 199: 0.5089
Epoch 200/200
90/90 [==============================] - 1s 8ms/step - loss: 0.1077 - accuracy: 0.9668 - val_loss: 3.8590 - val_accuracy: 0.5212

Test accuracy after epoch 200: 0.5091
```

In [134]:

```python
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.15,7/3))
plt.subplot(1, 2, 1)
plt.plot(history2_2.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history2_2.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback2_2.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history2_2.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights2_2 = []

for layer in model2_2.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights2_2.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 2:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 3:
            axes[row, col].set_title(f'Histogram of Dense 2')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
```

```
    plt.show()

plot_weights_histogram(all_weights2_2)

plt.show()
```



From the above plot, we can find that when we increse thestride size from (3,3) to (1,1), the results are similar to the previous model.

1. After 100 epochs, the learning rate tend to stabilize.
2. After 50 epochs,the validation accuracy and testing accuracy rates tend to stabilize around 0.5.

However, this model has more trainable parameters compared to the previous one, resulting in higher computational costs.

model2_3

Then, we try to set different filter size$=(2,2)$.

- Epochs = 200
- batch size = 500
- strides=(3,3),filter=(2,2)
- activation='relu','softmax'(output)
- Optimizer = Adam

In [135]:

```python
model2_3 = models.Sequential()

#The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model2_3.add(layers.Conv2D(64, (2, 2), strides=(3,3), padding='same', activation='relu', input_shape=(32, 32, 3)))
model2_3.add(layers.MaxPool2D(((2, 2))))
model2_3.add(layers.Conv2D(64, (2, 2), strides=(3,3),padding='same', activation='relu'))
model2_3.add(layers.MaxPool2D(((2, 2))))

model2_3.add(layers.Flatten())
model2_3.add(layers.Dense(384, activation='relu'))
model2_3.add(layers.Dropout(0.2))
model2_3.add(layers.Dense(192, activation='relu'))
model2_3.add(layers.Dropout(0.2))
model2_3.add(layers.Dense(10, activation='softmax'))
model2_3.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model2_3.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_41 (Conv2D)           (None, 11, 11, 64)        832       
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 2, 2, 64)          16448     
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 1, 1, 64)          0         
_________________________________________________________________
flatten_14 (Flatten)         (None, 64)                0         
_________________________________________________________________
dense_35 (Dense)             (None, 384)               24960     
_________________________________________________________________
dropout_21 (Dropout)         (None, 384)               0         
_________________________________________________________________
dense_36 (Dense)             (None, 192)               73920     
_________________________________________________________________
dropout_22 (Dropout)         (None, 192)               0         
_________________________________________________________________
dense_37 (Dense)             (None, 10)                1930      
=================================================================
Total params: 118,090
Trainable params: 118,090
Non-trainable params: 0
_________________________________________________________________
```

In [136]:

```python
from tensorflow.keras.callbacks import Callback

class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback2_3 = TestAccuracyCallback(test_data=(test_image2, test_label2))
```

In [137]:

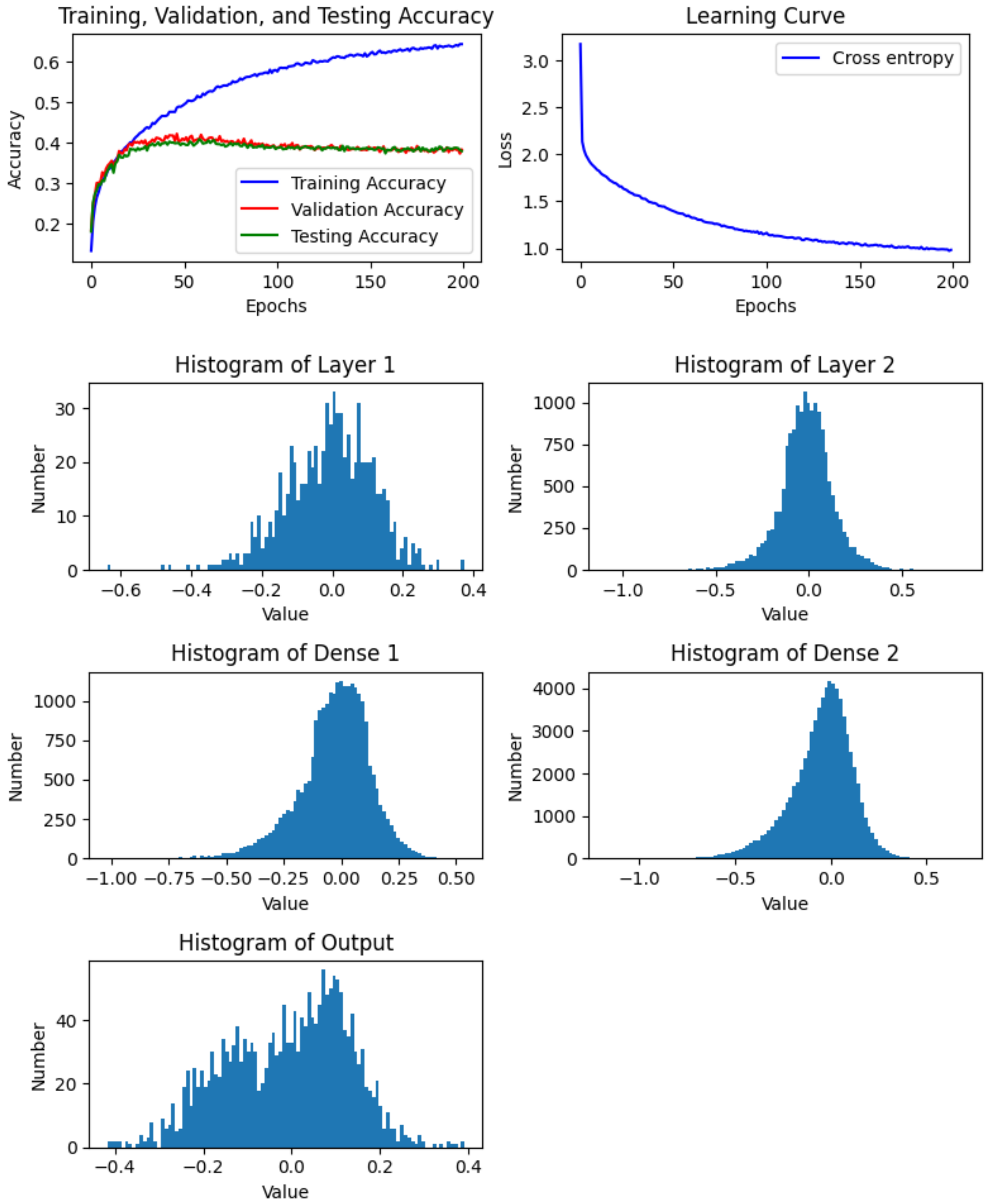
```python
history2_3 = model2_3.fit(train_image2, train_label2, batch_size=500, epochs=200, validation_split=5000/50000, callbacks=[test_callback2_3])
```

```
90/90 [==============================] - 0s 4ms/step - loss: 0.9914 - accuracy: 0.6376 - val_loss: 2.1377 - val_accuracy: 0.3858

Test accuracy after epoch 196: 0.3865
Epoch 197/200
90/90 [==============================] - 0s 5ms/step - loss: 0.9891 - accuracy: 0.6399 - val_loss: 2.1566 - val_accuracy: 0.3816

Test accuracy after epoch 197: 0.3869
Epoch 198/200
90/90 [==============================] - 0s 4ms/step - loss: 0.9901 - accuracy: 0.6378 - val_loss: 2.1518 - val_accuracy: 0.3824

Test accuracy after epoch 198: 0.3858
Epoch 199/200
90/90 [==============================] - 0s 5ms/step - loss: 0.9747 - accuracy: 0.6422 - val_loss: 2.1816 - val_accuracy: 0.3724

Test accuracy after epoch 199: 0.3811
Epoch 200/200
90/90 [==============================] - 0s 4ms/step - loss: 0.9845 - accuracy: 0.6422 - val_loss: 2.1788 - val_accuracy: 0.3822

Test accuracy after epoch 200: 0.3786
```

In [139]:

```python
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.15,7/3))
plt.subplot(1, 2, 1)
plt.plot(history2_3.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history2_3.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback2_3.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history2_3.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights2_3 = []

for layer in model2_3.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights2_3.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 2:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 3:
            axes[row, col].set_title(f'Histogram of Dense 2')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
```

```
    plt.show()

plot_weights_histogram(all_weights2_3)

plt.show()
```



From the above plot, we can find that when we increse the filter size from (5,5) to (2,2), the results are similar to the previous model.

1. After 50 epochs, the validation accuracy and testing accuracy rates tend to stabilize around 0.4.
2. The learning rate decreases slowly.

However, this model has less trainable parameters compared to the model 2_1, resulting in lower computational costs.

2-2 Show some examples of correctly classified and miss-classified images and discuss your results.

In [238]:

```
pred = model2_1.predict(test_image2)
pred_labels = np.argmax(pred, axis=1) #10000個

# true_and_pred_cat
correct_indices = np.where((test_label2 == 3) & (pred_labels == 3))[0]

# true_not_cat_pred_cat
wrong_indices = np.where((test_label2 != 3) & (pred_labels == 3))[0]

# true_and_pred_cat 第一張圖
plt.figure(figsize=(6, 3))
plot_label = plt.GridSpec(1, 2, wspace=0.5)
plt.subplot(1, 2, 1)
plt.imshow(test_image2[correct_indices[0]], cmap=plt.get_cmap('gray'))
plt.title(f'True: {label_names[test_label2[correct_indices[0]]]}\nPred: {label_names[pred_labels[correct_indices[0]]]}')
plt.axis('off')

# true_not_cat_pred_cat 第一張圖
plt.subplot(1, 2, 2)
plt.imshow(test_image2[wrong_indices[0]], cmap=plt.get_cmap('gray'))
plt.title(f'True: {label_names[test_label2[wrong_indices[0]]]}\nPred: {label_names[pred_labels[wrong_indices[0]]]}')
plt.axis('off')

plt.tight_layout()
plt.show()
```

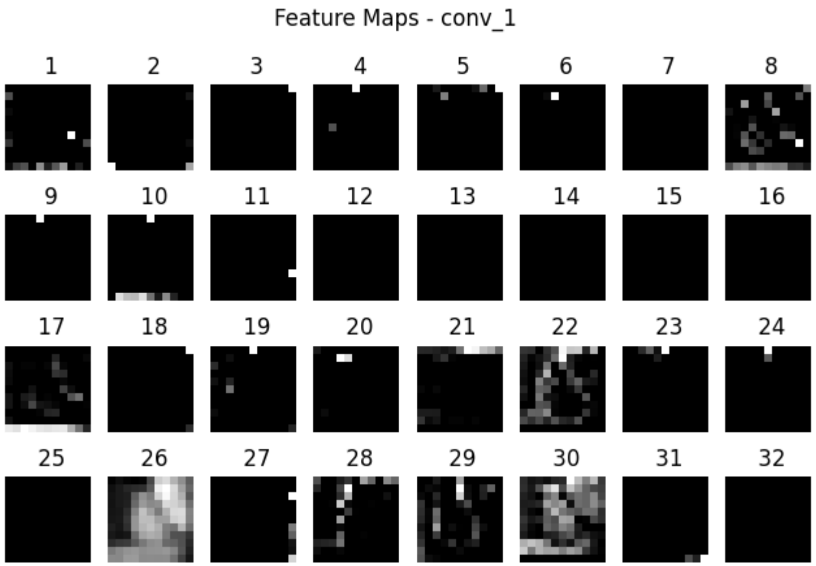
The above plots are examples of correctly classified and miss-classified images.
It show a true label and predicted label of cat and another example where the true label is not cat but the predicted label is cat.
I think the background of the truck picture has a pattern looks like cat ears, so it is miss-classified.

2-3 Following 2-2, observe the feature maps from different convolutional layers and describe how a feature map changes with increasing depth.

In [358]:

```
from tensorflow.keras.models import Model


# Plot the feature maps
plt.figure(figsize=(2, 2))

# Plot the original images for both cases

plt.imshow(test_image2[correct_indices[0]], cmap='gray')
plt.title(f'True: {test_label2[correct_indices[0]]}\nPred: {pred_labels[correct_indices[0]]}')
plt.axis('off')


# Choose an input image from the test set
input_image_1 = test_image2[correct_indices[0]].reshape(1,32,32,3)
input_image_2 = test_image2[wrong_indices[0]].reshape(1,32,32,3)

# Get feature maps for the last convolutional layer in each block
ixs = [0, 2]
outputs = [model2_1.layers[i].output for i in ixs]
model = tf.keras.Model(inputs=model2_1.input, outputs=outputs)
feature_maps_1 = model.predict(input_image_1)
feature_maps_2 = model.predict(input_image_2)
layer_names_2 = [layer.name for layer in model2_1.layers[:2] if 'conv2d' in layer.name ]

j = 1
for layer_name, fmap in zip(layer_names_2, feature_maps_1):
        plt.figure(figsize=(8, 5))
        for i in range(32):
            plt.subplot(4, 8, i + 1)
            plt.imshow(fmap[0, :, :, i], cmap='gray')
            plt.axis('off')
            plt.title(f' {i+1}')
        plt.suptitle(f'Feature Maps - conv_{j}')
        j += 1
        plt.show()
```

True: 3
Pred: 3

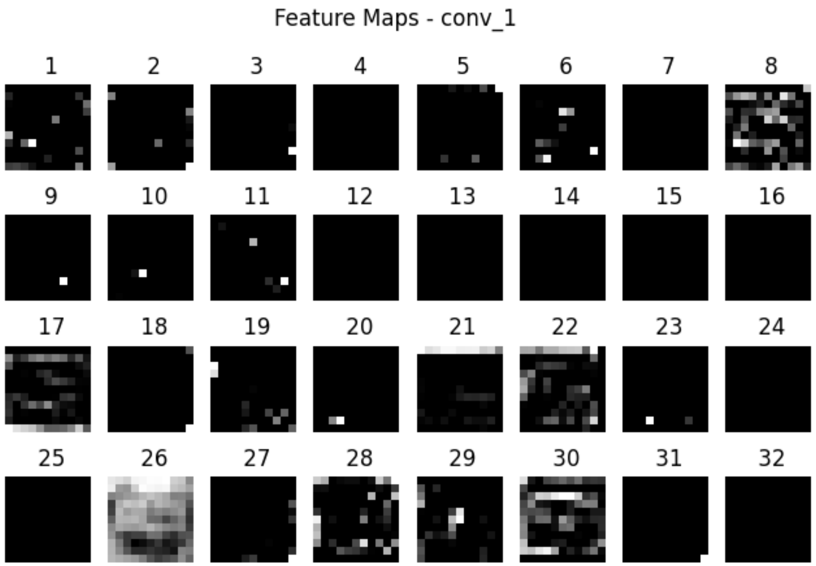


From the above plot, we can find that:

1. Feature maps of convolution layer1 capture some detail in the image, such as edges, corners, and textures. we can see the cat body Contour at the 22th plot.

In [359]:
```python
# Plot the feature maps
plt.figure(figsize=(2, 2))

# Plot the original images for both cases

plt.imshow(test_image2[wrong_indices[0]], cmap='gray')
plt.title(f'True: {test_label2[wrong_indices[0]]}\nPred: {pred_labels[wrong_indices[0]]}')
plt.axis('off')

j = 1
for layer_name, fmap in zip(layer_names_2, feature_maps_2):
        plt.figure(figsize=(8, 5))
        for i in range(32):
            plt.subplot(4, 8, i + 1)
            plt.imshow(fmap[0, :, :, i], cmap='gray')
            plt.axis('off')
            plt.title(f' {i+1}')
        plt.suptitle(f'Feature Maps - conv_{j}')
        j += 1
        plt.show()
```

True: 9
Pred: 3



From the above plot, we can find that:

1. Feature maps of convolution layer1 capture some detail in the image, such as edges, corners, and textures.
   We can see that the 22th, 30th plots look like a cat body Contour, so maybe that's the reason of being mis-classified.

2-4 Following 2-1, please add L2 regularization to the CNN implemented in 2-1 and discuss its effect.

We set $\alpha = 0.05$ of L2 regularization.

- Epochs = 200
- batch size = 500
- strides=(3,3),filter=(5,5)
- kernel_regularizer=0.05
- activation='relu','softmax'(output)
- Optimizer = Adam

In [326]:

```python
model2_L2 = models.Sequential()

#The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.
model2_L2.add(layers.Conv2D(64, (2, 2), strides=(3,3), padding='same', activation='relu', kernel_regularizer=regularizers.l2(0.05), input_shape=(32, 32, 3)))
model2_L2.add(layers.MaxPool2D(((2, 2))))
model2_L2.add(layers.Conv2D(64, (2, 2), strides=(3,3),padding='same', activation='relu', kernel_regularizer=regularizers.l2(0.05)))
model2_L2.add(layers.MaxPool2D(((2, 2))))

model2_L2.add(layers.Flatten())
model2_L2.add(layers.Dense(384, activation='relu' ,kernel_regularizer=regularizers.l2(0.05)))
model2_L2.add(layers.Dropout(0.2))
model2_L2.add(layers.Dense(192, activation='relu', kernel_regularizer=regularizers.l2(0.05)))
model2_L2.add(layers.Dropout(0.2))
model2_L2.add(layers.Dense(10, activation='softmax'))
model2_L2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model2_L2.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_43 (Conv2D)           (None, 11, 11, 64)        832       
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 2, 2, 64)          16448     
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 1, 1, 64)          0         
_________________________________________________________________
flatten_15 (Flatten)         (None, 64)                0         
_________________________________________________________________
dense_38 (Dense)             (None, 384)               24960     
_________________________________________________________________
dropout_23 (Dropout)         (None, 384)               0         
_________________________________________________________________
dense_39 (Dense)             (None, 192)               73920     
_________________________________________________________________
dropout_24 (Dropout)         (None, 192)               0         
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1930      
=================================================================
Total params: 118,090
Trainable params: 118,090
Non-trainable params: 0
_________________________________________________________________
```

In [327]:

```python
from tensorflow.keras.callbacks import Callback

class TestAccuracyCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        print(f'\nTest accuracy after epoch {epoch + 1}: {test_acc:.4f}')
        self.test_accuracy.append(test_acc)


test_callback2_L2 = TestAccuracyCallback(test_data=(test_image2, test_label2))
```

In [328]:

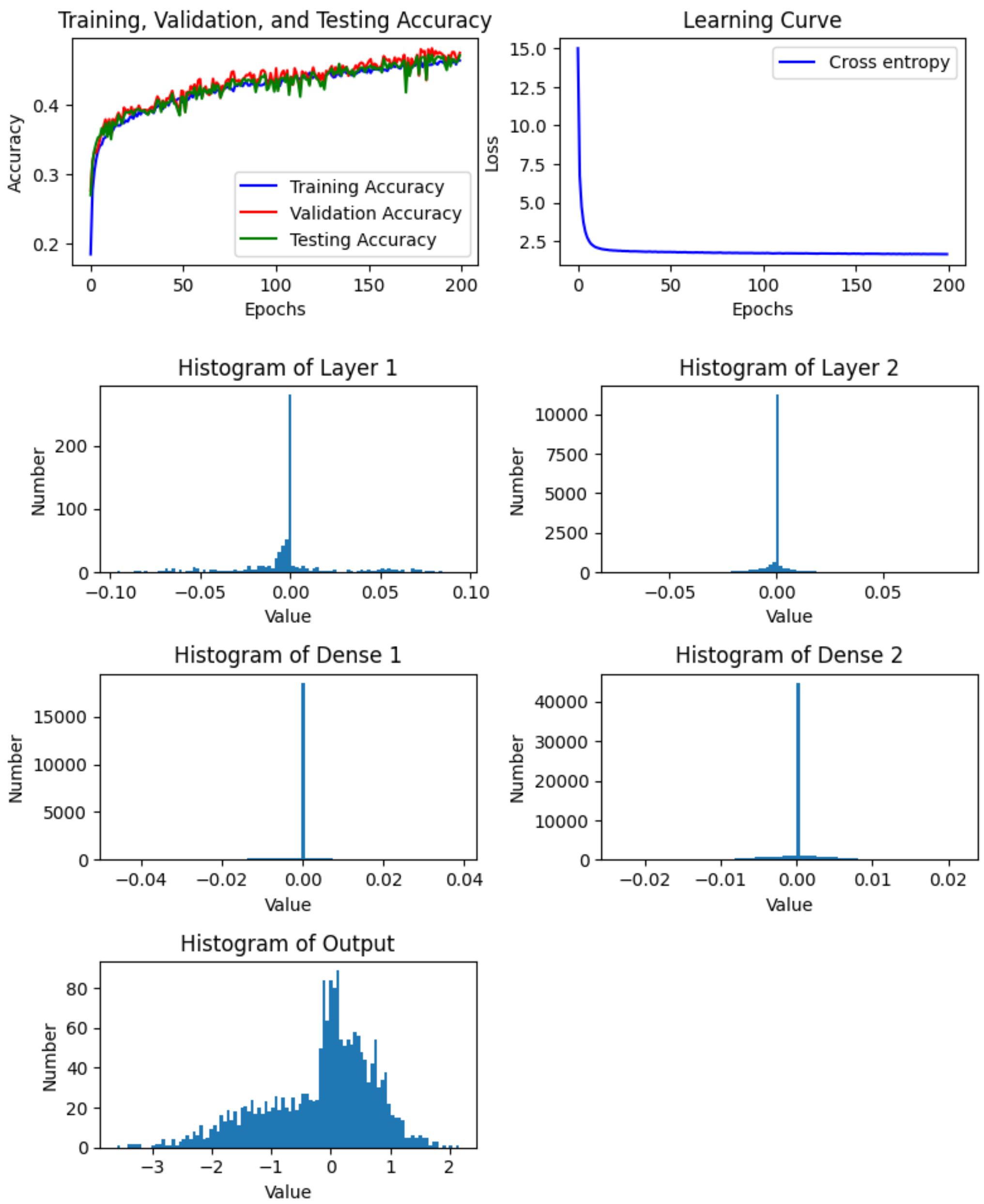
```python
history2_L2 = model2_L2.fit(train_image2, train_label2, batch_size=500, epochs=200, validation_split=5000/50000, callbacks=[test_callback2_L2])
```

```
Epoch 196/200
90/90 [==============================] - 0s 5ms/step - loss: 1.6772 - accuracy: 0.4600 - val_loss: 1.6435 - val_accuracy: 0.4754

Test accuracy after epoch 196: 0.4657
Epoch 197/200
90/90 [==============================] - 1s 6ms/step - loss: 1.6636 - accuracy: 0.4640 - val_loss: 1.6507 - val_accuracy: 0.4752

Test accuracy after epoch 197: 0.4637
Epoch 198/200
90/90 [==============================] - 0s 5ms/step - loss: 1.6742 - accuracy: 0.4575 - val_loss: 1.6643 - val_accuracy: 0.4666

Test accuracy after epoch 198: 0.4600
Epoch 199/200
90/90 [==============================] - 1s 6ms/step - loss: 1.6621 - accuracy: 0.4653 - val_loss: 1.6471 - val_accuracy: 0.4654

Test accuracy after epoch 199: 0.4626
Epoch 200/200
90/90 [==============================] - 1s 6ms/step - loss: 1.6680 - accuracy: 0.4632 - val_loss: 1.6399 - val_accuracy: 0.4742

Test accuracy after epoch 200: 0.4705
```

In [329]:

```
import seaborn as sns

# Plotting training, validation, and testing accuracy
plt.figure(figsize=(9.15,7/3))
plt.subplot(1, 2, 1)
plt.plot(history2_L2.history['accuracy'],color='blue', label='Training Accuracy')
plt.plot(history2_L2.history['val_accuracy'],color='red', label='Validation Accuracy')
plt.plot(test_callback2_L2.test_accuracy,color='green', label='Testing Accuracy')
plt.title('Training, Validation, and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting learning curve
plt.subplot(1, 2, 2)
plt.plot(history2_L2.history['loss'],color='blue', label='Cross entropy')
plt.title('Learning Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

## PLOT histograms
all_weights2_L2 = []

for layer in model2_L2.layers:
     # check if weight exists
    if layer.get_weights():
        weights, _ = layer.get_weights()  # ignore bias
        all_weights2_L2.append(weights.flatten())

# 绘制直方图的函数
def plot_weights_histogram(weights_list):
    num_layers = len(weights_list)

    # Calculate the number of rows and columns based on the number of histograms
    num_rows = (num_layers - 1) // 2 + 1
    num_cols = min(num_layers, 2)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 7))  # Dynamic grid size

    for i in range(num_layers):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index

        axes[row, col].hist(weights_list[i], bins=100, linewidth=1.2)

        if i == 2:
            axes[row, col].set_title(f'Histogram of Dense 1')
        elif i == 3:
            axes[row, col].set_title(f'Histogram of Dense 2')
        elif i == 4:
            axes[row, col].set_title(f'Histogram of Output')
        else:
            axes[row, col].set_title(f'Histogram of Layer {i + 1}')

        axes[row, col].set_xlabel('Value')
        axes[row, col].set_ylabel('Number')

    for i in range(num_layers, num_rows * num_cols):
        row = i // 2  # Integer division to determine the row index
        col = i % 2   # Modulus operation to determine the column index
        fig.delaxes(axes[row, col])

    plt.tight_layout()
```

```
    plt.show()

plot_weights_histogram(all_weights2_L2)

plt.show()
```



We can find that:

1. The accuracy rate of train, test, validation are closed.
2. The regularization term encourages the network to learn smaller weight values.
3. The histograms of weights appear to be concentrated around a narrow range, and the value appear to 0.