CS 482/682 Machine Learning: Deep Learning
Mock exam questions

Note: Please note that the primary purpose of this mock exam is for you to review your understanding of important concepts taught in this class. Specifically, the mock exam **is not** representative of the true midterm exam with respect to length or composition of questions (there will be no true/false or multiple choice questions.

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	10	12	25	26	8	4	25	28	18	156
Score:										

1. **True/False Questions**
   (a) (2 points) In standard recurrent neural networks, the magnitude of the largest singular value of the weight matrix $\boldsymbol{W}$ is related to the exploding/vanishing gradient problem.
   ◯ (2pt) True is correct, because gradient is proportional to W and the largest singular value defines a bound on the magnitude of increase/decrease after multiplication, i.e. gradient update
   (b) (2 points) When moving from fully connected multi-layer perceptrons to convolutional neural networks, a key assumption is translation invariance.
   ◯ (2pt) True is correct, because weights are shared across spatial dimension
   (c) (2 points) There is no obvious relation between convolutional layers and fully-connected layers.
   ◯ True ◯ False (2pt) FC layers can be expressed by conv layers with specific kernel size
   (d) (2 points) Linear regression with L_2 loss is convex and we can always find the globally optimal solution. ◯ True (2pt) correct, because L2 Hessian is positive semi-definite → convex ◯ False
   (e) (2 points) Perceptrons can model any Boolean function.
   ◯ True ◯ False (2pt) no XOR

2. **Multiple-choice Questions** (More than one answer can be correct!)
   (a) (4 points) Convolutional neural network (ConvNet) architectures ...
   ◯ Most of the parameters of VGG-16 are in the early convolutional layers
   (1pt) This is incorrect. For VGG, most parameters are in the first fully connected layers.
   ◯ Blocks of 3 3×3 convolutions with stride 1, as is done in VGG, mimic the 11×11 convolutions in AlexNet but has fewer parameters and more non-linearities
   (1pt) This is incorrect, because this filter combination has a receptive field of 7×7. However, fewer parameters and more non-linearities is correct.
   ◯ Residual connections, as in the ResNet, introduce "skip ahead connections" with "addition nodes" such that the gradient is distributed, enabling more effective optimization
   (1pt) This is correct.

◯ Convolutions with strides > 1 can be used instead of pooling layers
(1pt) This is correct. ResNet implements it this way.

(b) (4 points) Recurrent neural networks and LSTM ...

◯ Can model sequential data and are often used for natural language processing tasks (1pt) True

◯ Do not suffer from vanishing and exploding gradient problems (1pt) False

◯ Vanilla-RNN can learn to retain the most relevant parts of each frame (1pt) False

◯ LSTMs use the forget gate to extend the number of frames it can retain information over (1pt) True

(c) (4 points) You observe that your training loss initially decreased but does not converge to a stable solution. Which of the following may help you achieve convergence? Solution remarks: This suggests that training does not succeed. Since loss decreases initially, this is most likely an optimization problem

◯ Increasing the network depth (1pt) False, since this would yield to even more complicated optimization problem

◯ Increasing the batch size (1pt) True, since this would yield more stable gradients and the optimizer will "see" more of the cost function

◯ Increasing the learning rate (1pt) False, since this would likely make the optimization diverge further

◯ Increasing the dropout rate (1pt) True, since added regularization can aid optimization. Hand-waving: "adding curvature" to the cost function, disambiguation

3. **Convolutional Neural Networks: Concepts** In this problem you will try to reason why the sigmoid activation function in intermediate layers has largely been replaced by Rectified Linear Units (ReLU).

(a) (4 points) First, state the mathematical formula of the sigmoid $\sigma(x)$ and sketch (with annotations of y-axis crossing and limits) the curve in the interval $[-10, 10]$.

- (2pt) $\sigma(x) = \frac{1}{1+\exp(-x)}$
- (2pt) for plot. (1pt) for range $[0, 1]$, (1pt) for $\sigma(0) = 0.5$

(b) (4 points) The derivative of the sigmoid function $\sigma(x)$ is given by $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$. Plot the derivative into the same diagram. What are the minimum and maximum values of the derivative?

- (1pt) Minimum is 0
- (1pt) Maximum is 0.25
- (2pt) for plot. (1pt) for range $[0, 1]$, (1pt) for $d(\partial\sigma)(0) = 0.25$

(c) (4 points) Briefly explain the consequences of the previous observations for parameter updates during training of *very deep (or recurrent)* neural networks via backpropagation when using sigmoid as activation function?

- (1pt) Gradients are calculated using chain rule, local gradient multiplied with upstream gradient
- (1pt) Gradient magnitude through sigmoid layers is bounded and always ≤ 0.25
- (1pt) Gradient magnitudes always shrink when passed through sigmoid layers
- (1pt) This makes learning of shallow layers (close to input) difficult because gradient magnitude is small

(d) (5 points) A common alternative to the sigmoid activation function, that has contributed substantially to the rise of deep learning, is the rectified linear unit (ReLU) activation function. Give its formula, plot its curve, and briefly explain how ReLU solves the aforementioned problem of the sigmoid.

- (1pt) $\text{ReLU}(x) = \max(0, x)$
- (2pt) Plot; (1pt) $\text{ReLU}(0) = 0$, (1pt) line plot

- (2pt) (1pt) Mitigates vanishing gradient problem (1pt) because derivative is 1 for every $x > 0$

(e) (8 points) However, the ReLU is not ideal. First, give two examples of problems that remain or are introduced when using the ReLU. Second, introduce a small modification to ReLU that solves these problem, give it's formula, and plot it in the diagram of (d).

- (2pt) Problem 1: (1pt) Outputs are not zero-centered (but all positive) (1pt) which introduces ineffective gradient updates
- (2pt) Problem 2: (1pt) Dead ReLU problem (1pt) because gradient is always 0, unfortunate initialization may lead to some neurons never activating for the data set. Gradients will always be 0, the ReLU is "dead"
- (2pt) LeakyReLU$(x) = \max(\alpha x, x)$
- (2pt) Plot (1pt) zero crossing, (1pt) overall curve ReLU$(x) = \max(0, x)$

4. **Convolutional Neural Networks: Regularization** In this problem you will consider regularization approaches to avoid overfitting when training convolutional neural networks (ConvNets).

(a) (8 points) First, briefly explain the *bias – variance trade-off* and provide explanations on what we mean by bias and variance. Second, describe how this relates to optimizing the parameters of ConvNets.

- (1pt) Bias – variance trade-off refers to the observation that overall error in optimization can be decomposed into bias, variance, and irreducible residual error
- (2pt) Bias: (1pt) Bias describes a systematic error of the model that arises from a simpler, regularized model: (1pt) This introduces some "resistance" of the model to adapt to statistic outliers, but will favor models that may be too simplistic.
- (2pt) Variance: (1pt) Variance is the exact opposite and describes the desire of the model to explain all variation present in the observation (i. e. the training data set). (1pt) If the model allows for it, a model with high variance will be able to perform perfectly on the training data but will not generalize well since the model is too complicated.
- (3pt) Relation no ConvNets: (1pt) ConvNets are over-parameterized so they are at risk of "overfitting" (variance). (1pt) We must design regularization strategies (introduce bias) to find an optimal operating point in the bias – variance trade-off design space. (1pt) This is necessary to train models that perform and generalize well.

(b) (9 points) In class, we discussed batch normalization as a way of introducing regularization. We now consider the training phase. Using bullet points, please outline the procedure of applying batch normalization. Assume that your batch contains $i = 1, \ldots, N$ instances $\boldsymbol{x}_i^{(j)}$ with $j = 1, \ldots, M$ feature maps. Please provide 1) a formula describing the computations, and 2) a brief rationale on why this computation is performed.

- (3pt) Compute batch mean and standard deviation for every feature map j over all batched instances i: (1pt) description, (1pt) formula mean, (1pt) formula standard deviation
- (3pt) Apply normalization across the batch (again, over every feature map separately): (1pt) description, (1pt) mean subtraction, (1pt) normalization
- (3pt) Introduce trainable mean and standard deviation and shift output distribution according to learned parameters: (1pt) description, (1pt) variance re-scaling, (1pt) mean addition

(c) (2 points) How is batch normalization applied in the testing phase?

- (1pt) Mean and standard deviation for steps 1,2 above are estimated from the training data
- (1pt) They are then fixed during application

(d) (7 points) Dropout is another way of regularizing ConvNets. First, please briefly describe how dropout is implemented and why it regularizes ConvNets during training. Second, please briefly describe the change(s) that must be made for testing? Finally, can you find a reason to apply dropout during testing?

- (1pt) Dropout prevents feature co-adaptation
- (1pt) For a network with N neurons, dropout randomly samples over the 2^N sub-networks
- (1pt) All sub-networks must still perform the same task, so this is comparable to an ensemble method
- (2pt) During application: (1pt) Because now all neurons are *on* and contribute to computations, (1pt) either the weights or feature maps must be scaled proportionally to the dropout rate
- (2pt) Test-time dropout: (1pt) provides multiple different predictions for the same input, (1pt) allowing for an estimate of (un)certainty in the prediction

5. **Convolutional Neural Networks: Optimization** In this problem, you will analyze different optimization strategies that can be applied to find optimal parameters when training neural networks using gradient descent-type minimization, e. g. backpropagation. We seek to find $\hat{\boldsymbol{x}}$ such that $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} L(\boldsymbol{x})$, where $L(\boldsymbol{x}) : \mathbb{R}^2 \mapsto \mathbb{R}$ is an arbitrary multi-dimensional scalar-valued function (function accepts vector and yields scalar), e. g. a loss function.

   (a) (8 points) Consider the following notation. At iteration t, where $t \gg 0$, we have:

$$\begin{aligned} \text{Gradient:} \quad & \boldsymbol{g}_t = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_t) \\ \text{?:} \quad & \\ \text{Element-wise update:} \quad & (\mathrm{d}\boldsymbol{x}_t)_i = \\ \text{Update step:} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \mathrm{d}\boldsymbol{x}_t \end{aligned}$$

   where $(\boldsymbol{x}_t)_i$ corresponds to the i-th element of $\boldsymbol{x}$. Please provide the missing computations for SGD with momentum, and RMSProp.

   SGD with momentum

$$\begin{aligned} \text{Gradient:} \quad & \boldsymbol{g}_t = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_t) \\ \text{?:} \quad & \boldsymbol{v}_{t+1} = \rho \boldsymbol{v}_t + \alpha \boldsymbol{g}_t \text{ (2pt)} \quad \text{with } (\boldsymbol{v}_t)_i(t=0) = 0 \text{ (1pt)} \\ \text{Element-wise update:} \quad & (\mathrm{d}\boldsymbol{x}_t)_i = (\boldsymbol{v}_{t+1})_i \text{ (1pt)} \\ \text{Update step:} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \mathrm{d}\boldsymbol{x}_t \end{aligned}$$

   RMSProp

$$\begin{aligned} \text{Gradient:} \quad & \boldsymbol{g}_t = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_t) \\ \text{?:} \quad & S_i = \rho \cdot S_i + (1-\rho)(g_t)_i^2 \text{ (2pt)} \quad \text{with } S_i(t=0) = 0 \text{ (1pt)} \\ \text{Element-wise update:} \quad & (\mathrm{d}\boldsymbol{x}_t)_i = \frac{\alpha}{\sqrt{S_i + \epsilon}} (\boldsymbol{g}_t)_i \text{ (1pt)} \\ \text{Update step:} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \mathrm{d}\boldsymbol{x}_t \end{aligned}$$

6. **Convolutional Neural Networks: Architecture**

   One important consideration when designing convolutional neural network architectures is the *receptive field*. Large receptive fields can be achieved via large kernels, but there is another solution. Consider a convolutional kernel of size 7×7 that obviously has a receptive field of 7×7 pixels. Please briefly describe an alternative way of achieving the same receptive field (Hint: Think about the VGG architecture). Name and explain one advantage of the alternative method you just described.

   (a) (4 points)
   - (2pt) The same receptive field of 7×7 pixels can be achieved by (1pt) 3 consecutive (1pt) 3×3 convolutional layers
   - (2pt) Advante: (1pt) More convolutional layers introduce more (1pt) non-linearities

7. **Multi-layer perceptron** In this question, you will study the capacity of multi-layer perceptrons (MLPs). Consider the dataset shown in Fig. 1 that was sampled from a 2D distribution. Note how the decision boundary that optimally separates these 2D points can be modeled by two simple polygons. In

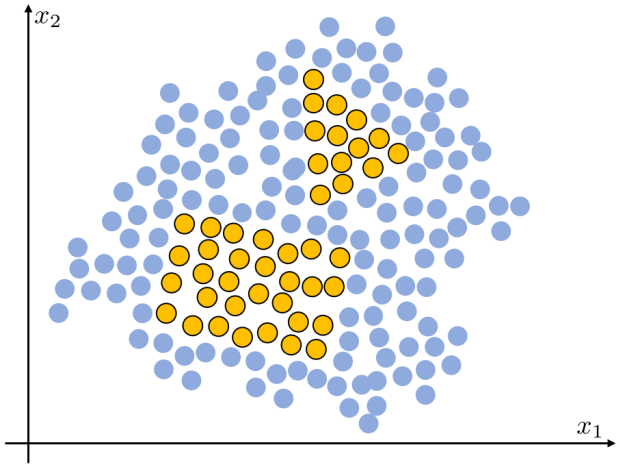

Figure 1: Visualization of an example 2D dataset with true/false annotation. For mathematical modeling, *true* corresponds to an output of 1 while *false* corresponds to an output of 0. Every instance $\boldsymbol{x}_i = (x_1^{(i)}, x_2^{(i)})$ is visualized as circle and colorized depending on the respective annotation: Instances are colored in blue if *false* and colored in orange with black outline if *true*.

the following exercise, you will design an MLP architecture that exploits this observation to classify data sampled from this distribution. Here, we consider perceptrons of model $f(\boldsymbol{u}, \boldsymbol{W}, b) = \begin{cases} 1, & \text{if } \boldsymbol{W}\boldsymbol{u} > b \\ 0, & \text{else} \end{cases}$ (i. e. threshold activation).

(a) (7 points) As we learned in class, every perceptron over the inputs $\boldsymbol{x}$ defines a linear decision boundary. Start by drawing all required linear decision boundaries on Fig. 1 and label them (e.ġ. $L_1, \ldots, L_n$). Please label the positive direction of the boundary (i.e. $f(x) > 0$).

- 0.5 points per line
- 0.5 points per direction (direction vector must point towards *true* class)

(b) (8 points) Using the above result, draw the computational graph of an MLP with two hidden layers that classifies the dataset. Make use of your annotations in Fig. 1 (e.ġ. $L_1, \ldots, L_n$) to link your computational graph with your drawings.

- 0.5 points perceptron / line in first layer (3.5 total)
- 2 points per perceptron in second layer → Perceptrons are fully connected. One point lost if connection only to lines corresponding to the polygon. (4 total)
- 05. points for last perceptron

(c) (10 points) By design, the previous MLP has two hidden layers. However, we have seen in class that the same (and any arbitrary decision boundary) can be approximated using an *MLP with a single hidden layer*. Please provide a brief walk-through on how this can be achieved (6 bullet points maximum!). You can use small schematic drawings to support your reasoning.
The below answer is approximately what we want to hear. There is some leniency to be applied if the answer does not fully follow the below, but the main concepts as per this answer must be provided.

- (1pt) A single perceptron in the first hidden layer defines a linear decision boundary
- (1pt) Linear decision boundaries can be arranged in 2D space such that, when *AND* together for the output, their decision boundary forms a closed polygon
- (2pt) The threshold for this MLP to fire only within this polygon is N, where N is the number of neurons used to build the polygon

- (2pt) In the limit of infinitely many neurons $N \to \infty$, the perceptrons can be selected such that the resulting decision boundary approximates an infinitely small circular decision boundary
- (2pt) This circle has a response of N within and $\frac{N}{2}$ everywhere else. This can be corrected with bias term of $-\frac{N}{2}$ such that the response within the circle is $\frac{N}{2}$ and 0 everywhere else.
- (2pt) We can now create infinitely many more such circles by adding, for every circle, infinite neurons to the first hidden layer. Because the response is 0 everywhere outside the circles, the final output neuron needs not change.

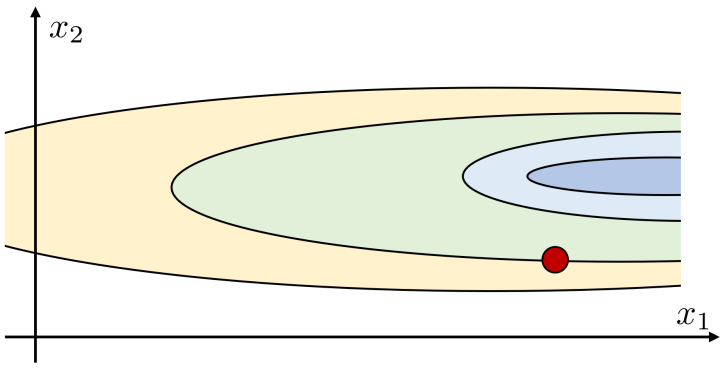

Figure 2: Contour lines of an arbitrary cost function $L(\boldsymbol{x}) : \mathbb{R}^2 \mapsto \mathbb{R}$. The function is convex in the displayed area with larger ellipses representing contour lines of greater values of $L(\boldsymbol{x})$. The red circle represents the current estimate $\boldsymbol{x}_t$ and we attempt to find the $\hat{\boldsymbol{x}}$ that minimizes L. *Please add your plots to this figure.*

8. **Convolutional Neural Networks: Optimization** In this problem, you will analyze different optimization strategies that can be applied to find optimal parameters when training neural networks using gradient descent-type minimization, e. g. backpropagation. We seek to find $\hat{\boldsymbol{x}}$ such that $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} L(\boldsymbol{x})$, where $L(\boldsymbol{x}) : \mathbb{R}^2 \mapsto \mathbb{R}$ is an arbitrary multi-dimensional scalar-valued function (function accepts vector and yields scalar), e. g. a loss function, shown in Fig. 2. Note, how values of the function change much more drastically in one direction compared to the other. The value at contour lines (lines on which $L(x)$ has the same value) is proportional to the size of the ellipse (i. e. *smallest* ellipse has *smallest* value!). **After multiple optimization steps**, the optimization algorithm has arrived at $\boldsymbol{x}_t$, that is highlighted as red circle in Fig. 2.

   (a) (8 points) Name and describe 2 problems of optimization with vanilla SGD and name and describe 2 approaches to mitigate them.

   - (2pt) Problem 1: (1pt) 0 gradients at local minima and saddle points; (1pt) optimization can stop early because gradient vanishes
   - (2pt) Problem 2: (1pt) Gradients can be noisy; (1pt) because true gradient is approximated from a random batch of data, updates can be ineffective particularly in poorly conditioned loss surfaces, e. g. "taco shell"
   - (2pt) Solution 1: (1pt) First order momentum; (1pt) Rather then updating with the gradient, keep "discounted running average" of previous gradients as velocity and merge with current, local gradient for the update
   - (2pt) Solution 2: (1pt) Second order momentum; (1pt) Keep "discounted running average" of gradient magnitudes with respect to every parameter separately. Then, use this second order moment to compute a parameter-wise adjustment to the learning rate

(b) (8 points) Consider the following notation. At iteration t, where $t \gg 0$, we have:

$$\begin{aligned} \text{Gradient:} \quad & \boldsymbol{g}_t = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_t) \\ \text{?:} \quad & \\ \text{Element-wise update:} \quad & (\mathrm{d}\boldsymbol{x}_t)_i = \\ \text{Update step:} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \mathrm{d}\boldsymbol{x}_t \end{aligned}$$

where $(\boldsymbol{x}_t)_i$ corresponds to the i-th element of $\boldsymbol{x}$. Please provide the missing computations for SGD with momentum, and AdaGrad.

SGD with momentum

$$\begin{aligned} \text{Gradient:} \quad & \boldsymbol{g}_t = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_t) \\ \text{?:} \quad & \boldsymbol{v}_{t+1} = \rho \boldsymbol{v}_t + \alpha \boldsymbol{g}_t \text{ (2pt)} \quad \text{with } (\boldsymbol{v}_t)_i(t=0) = 0 \text{ (1pt)} \\ \text{Element-wise update:} \quad & (\mathrm{d}\boldsymbol{x}_t)_i = (\boldsymbol{v}_{t+1})_i \text{ (1pt)} \\ \text{Update step:} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \mathrm{d}\boldsymbol{x}_t \end{aligned}$$

AdaGrad

$$\begin{aligned} \text{Gradient:} \quad & \boldsymbol{g}_t = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_t) \\ \text{?:} \quad & S_i = S_i + (\boldsymbol{g}_t)_i^2 \text{ (2pt)} \quad \text{with } S_i(t=0) = 0 \text{ (1pt)} \\ \text{Element-wise update:} \quad & (\mathrm{d}\boldsymbol{x}_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon} (\boldsymbol{g}_t)_i \text{ (1pt)} \\ \text{Update step:} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \mathrm{d}\boldsymbol{x}_t \end{aligned}$$

(c) (12 points) Now consider the situation in Fig. 2. In that figure, please plot the next update directions for vanilla SGD, SGD with momentum, and AdaGrad (do not consider the learning rate, i. e. arrows should be legible). For every optimizer, you should only draw **a single arrow**. Please also provide a brief statement in the space provided below explaining why you have decided to draw the update in this specific way.

- (4pt) SGD
  - (3pt) Statement: (1pt) SGD only considers local gradient. (1pt) The local gradient can be extracted from the drawing via the tangent to the contour line. (1pt) Local gradient is then orthogonal to the tangent.
  - (1pt) Drawing: Arrow is drawn correctly
- (4pt) SGD with momentum
  - (3pt) Statement: (1pt) Velocity is well initialized (after multiple updates), contributions in highly curved cost function direction have cancelled out. (1pt) Velocity is strongly aligned with true gradient direction. (1pt) Momentum update will be a weighted combination between local gradient (SGD) and velocity.
  - (1pt) Drawing: Arrow is drawn correctly
- (4pt) AdaGrad
  - (3pt) Statement: (1pt) Second order momentum term is well initialized (after multiple updates), contributions in highly curved cost function direction have accumulated making second order momentum large in that direction. (1pt) Second order momentum is much weaker in other direction. (1pt) Update will be taken with a parameter depend step size: strong dampening in steep direction, weak dampening in shallow direction. The update direction will the one of SGD. Due to adaptive weighting, step will be approximately equally long in both directions. The update direction will be at a 45° angle
  - (1pt) Drawing: Arrow is drawn correctly

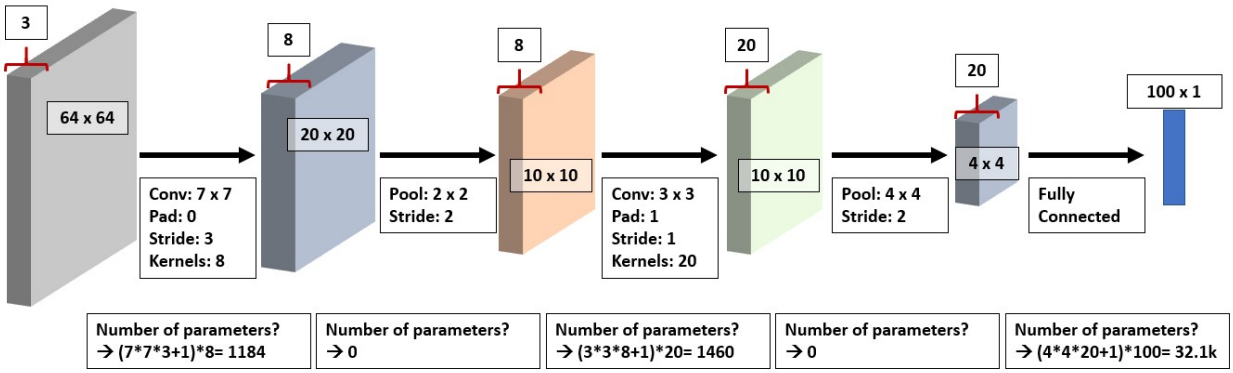

Figure 3: A shallow convolutional neural network for image classification. Your task is to derive the missing quantities highlighted as ??. Note: You will calculate the number of parameters in part (b).

9. **Convolutional Neural Networks: Architecture** Consider the network architecture provided in Fig. 3. The final output is a vector of size 100×1, e. g. for image classification in a problem with 100 different classes. Assume *ReLU activation* for all convolutional and fully connected layers.

   (a) (8 points) Several quantities (shown with question marks as ??) are missing from the schematic in Fig. 3. Please derive them from the provided values. If you write them directly into the figure make sure it is obvious (e. g. using arrows) which ?? they belong to.

   - (1pt) per missing value

   (b) (4 points) Now, please show the calculation for the number of parameters per layer and the total number of parameters. It is enough to write the formulae for the number of parameters; it is not necessary to perform the calculations to get a final number.

   - (1pt) per missing value (pooling counted only once)

   (c) (6 points) In class, we discussed the counter-intuitive observation that very deep networks (e. g. with 50 or more layers) that follow the simple structure shown in Fig. 3 do not necessarily outperform shallower networks. Please provide the reason for this phenomenon, name the observations that led to this conclusion, and *briefly* describe a way of overcoming this problem. To help your argumentation, sketch the key changes to the architectural setup.

   - (2pt) First observation: (1pt) Very deep networks performed poorly compared to shallower ones on **test set** → (1pt) This is overfitting problem
   - (2pt) Second observation: (1pt) Very deep networks performed poorly compared to shallower ones also on **training set** → (1pt) This is **not** overfitting, it is an optimization problem!
   - (1pt) Conceptually, deeper networks should be able to perform at least as well as shallower ones
   - (2pt) Add "residual" connections that (1pt) model identity and (2pt) improve gradient flow
   - (2pt) Drawing: (1pt) residual connection, (1pt) for labeling